Inside X68000

桒野雅彦 *Masahiko Kuwano* ● 著

Inside X68000

桒野雅彦 Masahiko Kuwano ● 著

初代 X 68000 があのグラディウスのテーマに乗って登場してきたのは 1987 年のことになります。国内では，PC-9801 の一人勝ちがほぼ確定し，その他のメーカもすべて 80X86＋MS-DOS となってしまい，パーソナルコンピュータの「パーソナル」が個人ユーザではなく，会社の中の各社員を指すだけのものになり下がってしまった，そんな時代であったと思います。シャープが X 1 の 16 ビット版を出すという話が流れたときも，「どうせ 86 系のマシンさ」「16 ビットは 98 でいいじゃないか」，そんな声が出てきてしまうほど，個人ユーザがパーソナルコンピュータに冷めてしまっていたようです。

そのようなユーザの前に現れた X 68000 は，これまでのパーソナルコンピュータに対するイメージを大きく転換させるものでした。縦型のスリムなデザインの中には個人ユーザが望んでいた CPU，68000 が搭載され，標準で 1 MB，最大 12 MB ものリニアなメモリ空間，65536 色のグラフィック，768×512 のビットマップのテキスト画面，スプライト，FM 音源，ADPCM，オートイジェクト機構付き 5 インチ FDD，3 D スコープ，画像取り込み，トラックマウス，HDD インタフェース標準装備……。予想すらしなかったその仕様と，40 万円を軽く切ってしまったその安さに，声も出なかった覚えがあります。

ワープロであったり，表計算機であったりする側面だけに目が向けられてしまい，ビジネス用の環境以外はすべてオプションとして買い揃えていくよりほかない「パソコン」と，何かを行いたくなったときに，すぐその場で試したり，考え方やイメージをその場で表現する欲求にすぐ応えてくれるだけのポテンシャルを持った「パーソナルコンピュータ」は，まったく別ものであるという考えから「パーソナルワークステーション」という言葉が出てきたのも当然でしょう。

パーソナルワークステーション，つまり個人の発想や直感に応え，それを表現し，実現し，さらなる発想に結び付けるためのプラットフォーム，5 年の歳月を経てもなお新しい X 68000 をその内側まで使いこみたいと思った方に，本書はきっとよき道案内となることでしょう。

1992 年 2 月　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　桒野雅彦

X68000の系譜

初代機以来，多くの機種が登場してきたX 68000シリーズを年代順に整理してみました。X 68000は新機種といっても，CPUやクロック周波数などをいたずらに変更するのではなく，ソフトウェア，ハードウェアの互換性を最大限に保ちつつ高集積化をはかり，空いたスペースをハードディスク（HDD）やSCSIやコプロセッサなど，従来，外付けユニットやオプションボードで対応していたものを内蔵できるようにしていくという方向に向いています。このような方針のため，たとえ初代機（無印）であっても，まったく古さを感じさせません。

1987年，初代X 68000が，翌年，集積度を向上させて20 MBの3.5インチHDDを内蔵できるようにしたX 68000 ACE/ACE-HDが登場します。

さらに1989年には，内蔵HDDの容量を40 MBまで上げたEXPERT，横型のPROがラインアップに追加されました。PROの系統は，従来のX 68000の系列のデザイン重視型とは異なり，ややビジネス臭さを感じさせるシリーズです。当然，ソフトウェア的には完全互換ですが，拡張スロットは縦型機の2スロットに対して4スロットと拡充され，マウスはトラックマウスではない，ごく普通のタイプになりましたし，キーボードはシリンドリカルステップスカルプチャ型でパームレストもあるようなものに変更されました。また，あまり利用されていなかった3Dスコープ端子は取り外されています。組み立てやすくなったためか，価格は縦型よりも低く抑えられていました。

翌1990年は，X 68000にとっては混沌の年ともいえるでしょう。EXPERT，PROシリーズにそれぞれ後継機が出たほかに，HDDインタフェースをSCSIに変更し，80 MBのHDDを内蔵させたSUPER-HDが追加されました。この年，一気に5機種が発売されたことになります。その後，SUPER-HDのHDDがないタイプであるSUPERが投入され，型番上も実質的にも，EXPERTの後継機となりました。

明けて1991年，初代機以来変更のなかったクロック周波数が10 MHzから16 MHzに引き上げられ，内部でのメモリ拡張性の向上，オプションボードで対応していたコプロセッサを本体内部に取り付けられるようにするなどの改良が加えられました(XVI)。

SUPERでHDDインタフェースがSCSIになったり，XVIでクロックが引き上げられたために動作が速くなったり，といった違いはありますが，ソフトウェアからみたときにはどの機種であっても完全な互換性を保っています。上位互換ということではありませんから，ある機種でつくったプログラムがそれ以前の機種では動作しないといったこともまず起こりません(初期のころはメモリを1 MBしか積んでいなかったため，2 MBあることを前提にしているソフトウェアが動かないということはあるでしょうが，これとてメモリを増設してあればすむことです)。

図……1 X68000シリーズの系譜

上段：機種名
下段：型番

1987年
(無 印)
CZ-600CE

20MBHDD内蔵

・横型
・拡張スロット数4
・3Dスコープ端子削除

1988年
ACE
CZ-601C

ACE-HD
CZ-611C

40MBHDD内蔵

40MBHDD内蔵

1989年
EXPERT
CZ-602C

EXPERT-HD
CZ-612C

PRO
CZ-652C

PRO-HD
CZ-622C

80MBHDD内蔵
SCSI化

1990年
EXPERT II
CZ-603C

SUPER-HD
CZ-623C

EXPERT II-HD
CZ-613C

PRO II
CZ-653C

PRO II-HD
CZ-663C

HDDインタフェース
SCSI化

SUPER
CZ-604C

CPUクロック16MHz

CPUクロック16MHz

1991年
XVI
CZ-634C

XVI-HD
CZ-644C

1992年
Compact
XVI
CZ-674C

本書では，I/Oの割り付けやブロック図などは，すべて初代機（無印）にもとづいて説明しており，またサンプルプログラムの動作チェックも初代機に増設メモリやコプロセッサボード，SCSIインタフェースボードなどを追加して行いました。念のため，本書の執筆時に使用していたシステムの構成を掲げておきます。

●システム#1

X 68000(初代機)＋内部増設(1 MB)＋拡張メモリボード(2 MB)＋コプロセッサボード(CZ-6BP1)＋40 MB-SASIハードディスク（キャラベル/H540S)＋カラーイメージユニット（CZ-6VT1)

●システム#2

X 68000 (初代機)＋内部増設 (1 MB)＋SCSIインタフェースボード (CZ-6BS1)＋100 MB-SCSIハードディスク（アイテック/TX-100)

ディスプレイはどちらもCZ-600 DEを使用。

サンプルプログラムについて

アセンブラよりもロジックが見やすく，流用や改造なども容易であろうということから，本書ではサンプルプログラムの記述にC言語を使用することにしました。

X 68000用のCコンパイラは，シャープ純正のXCとフリーソフトウェアのgccが広く出回っています。純正という意味ではXCを標準とすべきかもしれませんが，gccのほうが圧倒的に生成されるコードの質がよく，バグが少ないようですし，入手についても，パソコン通信や電脳倶楽部，ソフトバンクの『Cマガジン』や『Oh！X』誌の付録ディスクなど，多くのルートで配布されたことから，gccのほうが一般的ではないかと考え，これを採用しました。

このため，本書ではgccを標準としてプログラムを作成しています。一応，XCでもコンパイルして動作は確認していますが，このときには# define volatileの1行を入れるのを忘れないようにしてください。

また，サンプルプログラムはすべて割り込みを使用せず，ステータスチェックでぐるぐる回りながら動作の終了を待つ，というようにしています。X 68000は割り込みが有効に使えるようなハードウェアになっていますし，基本的に入出力動作などは割り込みで行うのが普通ですが，サンプルプログラムは動作確認がおもな目的ですし，割り込みベクタをいじり間違えたりすると，すぐに妙な動作を始めてしまうことになることから，このような方法をとりました。実際にアプリケーションをつくるような場合には，できるだけ割り込みを使うようにしたほうがよいでしょう。

サンプルプログラム目次

DMA 関連

数値演算プロセッサ関連

RTC 関連

画面制御関連

ADPCM 関連

FDD 関連

ハードディスク関連

CONTENTS

プリンタ …… 371

ジョイスティック …… 377

フロッピーディスクドライブ …… 387

SCSI

システムポート

COVER DESIGN……Masaki KATSUMATA

メモリマップ

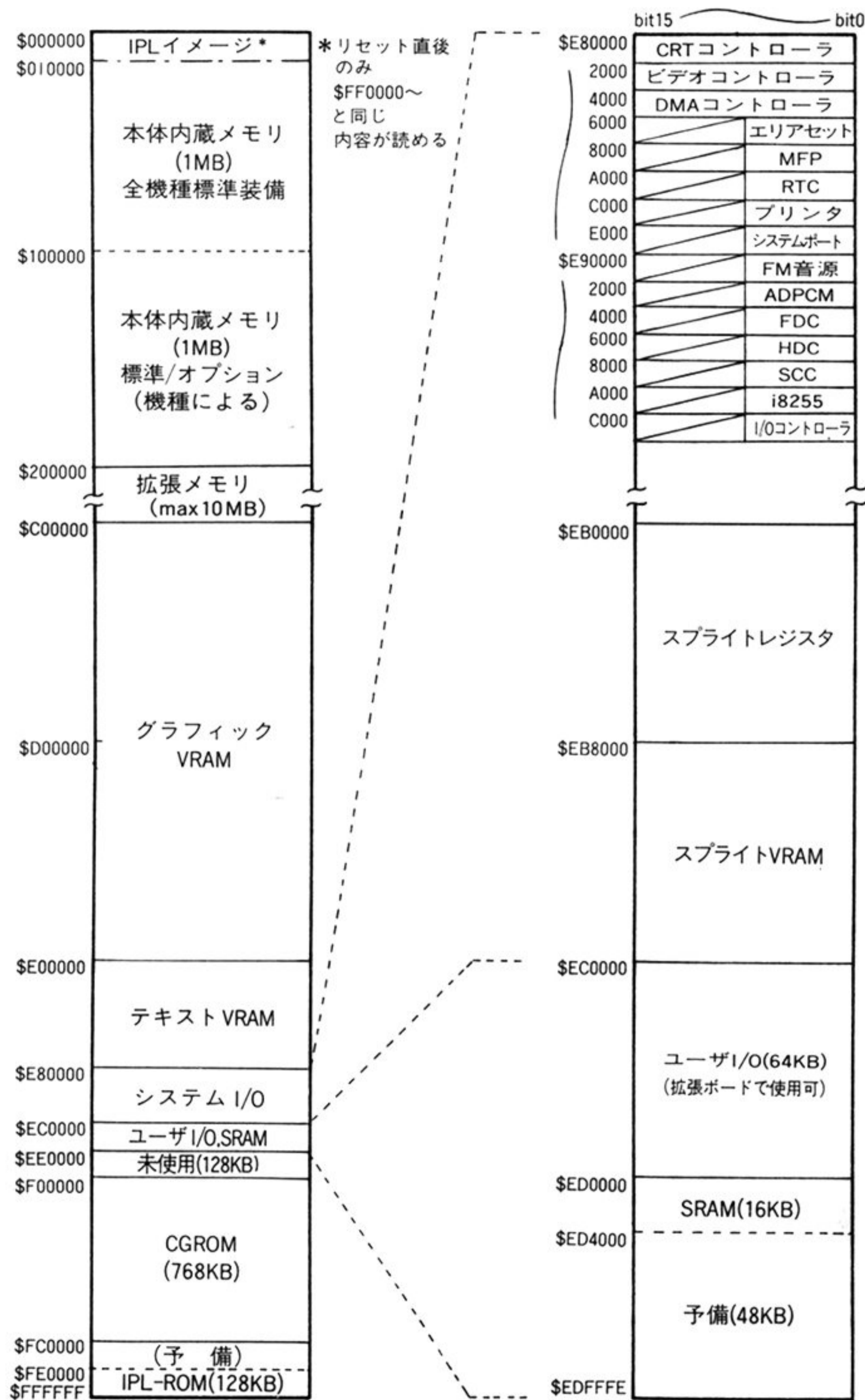
X68000 の CPU である 68000 には 80X86 のような I/O 空間はなく，16 M のメモリ空間があるだけです。ここでは X68000 で，このメモリ空間をどのように割り振っているかについて説明します。

1 メモリマップ

X 68000 のメモリマップを 20 ページの図 1 に示します。CPU の持つ 16 M バイトのメモリ空間のうち，0 番地から$BFFFFF までの 12 M バイト分がメインメモリの領域，$C 00000 以降がグラフィック画面やテキスト画面の VRAM や I/O，IPL-ROM の領域となっています。

2 IPLイメージ

68000 という CPU はリセットが解除されると，$000000 番地と$000004 番地から SSP(スーパーバイザスタックポインタ) と PC (プログラムカウンタ) の初期値を読み出して，動作を開始します。X 68000 の場合，0 番地側はメインメモリ領域となっていますので，何も細工を

●図……1　X68000のメモリマップ

しないと，CPUはリセット直後にDRAM上の不定のデータを読み出し，暴走してしまいます。そこで，X 68000では$000000～$00FFFFの64 Kバイトの領域は，電源投入直後やリセットスイッチによるリセット直後にかぎり，IPL-ROM領域の$FF0000～$FFFFFFの領域がそのまま見え(どちらからアクセスしてもROMの同じ領域が読める)，$FF0000～$FFFFFFの領域がアクセスされると，この領域がDRAM領域に切り替わるようにしています。この機構は，電源ONやリセットスイッチによるリセットがかかったときだけ働くようになっており，RESET命令などを実行しても，0番地からIPL-ROMの内容が読めるようにはなりません。

●3 メインメモリ

X 68000は最大12 Mバイトのメインメモリを持つことができます。この領域のうち，0番地からの1Mバイト分は，初代機以来すべての機種で標準装備されています。$100000から$1FFFFFまでの1Mバイト分は，標準で搭載しているものとオプションになっているものとがありますが，オプションに設定されているものであっても，本体内部で増設できるようになっています。

$200000番地以降の分の増設は，XVI以外の機種では拡張スロットにメモリボードを差し込んで行います。XVIは本体内部で8Mバイトまで増設できるようになっています。

●4 グラフィックVRAM

グラフィックVRAMは$C00000～$DFFFFFまでの2Mバイト分の空間がありますが，実際に搭載されているメモリは512 Kバイトです。X 68000のグラフィック画面は16色モード，256色モード，65536色モードの3種類がありますが，いずれの場合にも1ドットに1ワード分の領域がとってあります。16色や256色モードの場合には，1ワードのうち，下位の4ビット/8ビットだけが使用されるようになっています。このため，実際には512 Kバイトしかメ

モリがなくても，メモリ空間は実画面の最大サイズ，1024×1024 ドット分あるわけです。

●5 テキストVRAM

テキスト VRAM は 512 K バイト分が実装されています。テキスト画面は，1024×1024 ドットの画面が 4 プレーンという構成になっており，グラフィック画面のように無効なビットがないため，メモリ空間上も 512 K バイト分となっています。

●6 システム I/O領域

システム I/O 領域には，CRT や FD，FM 音源などの周辺機器制御用のデバイスや，スプライト用のメモリなどが配置されています。

●7 ユーザ I/O, SRAM

ユーザオリジナルの拡張ボードなどで使用できる領域として，$EC0000～$ECFFFF の 64 K バイト分が割り当てられています。この領域はユーザが自由に使用でき，アクセスもユーザモードから行うことができるようになっています。

SRAM はバッテリバックアップされているメモリで，電源を切っても，内容が保持されています。搭載されているメモリのサイズや画面の色の初期値など，システムのセットアップ用のデータを保存するなどの用途に使用されています。

8 CGROM

CGROM（キャラクタジェネレータ ROM）は，英数字，漢字などの文字のパターンが書き込まれているメモリです。X 68000 のテキスト画面はビットマップ方式であり，一種のグラフィック画面ですから，任意の文字パターンを表示できます。CGROM の中には 8 × 8，8 ×16，12×12，12×24 のドット構成の英数字と，16×16，24×24 ドット構成の漢字の，計 6 種類の文字パターンが用意されています。

9 IPL-ROM

CPU がリセット直後から実行するプログラムを書き込んでおく ROM です。X 68000 では空き番地に基本的な入出力サブルーチン（IOCS）などが収められています。Human 68 K の初期のバージョンでは，この ROM 内の IOCS を利用していましたが，現在は内容をより洗練した IOCS.X などを RAM 上に読み込んで，そちらを使うようになったため，IPL-ROM は周辺デバイスの基本的な初期設定と FD や HD からの起動処理程度にしか使われていません。

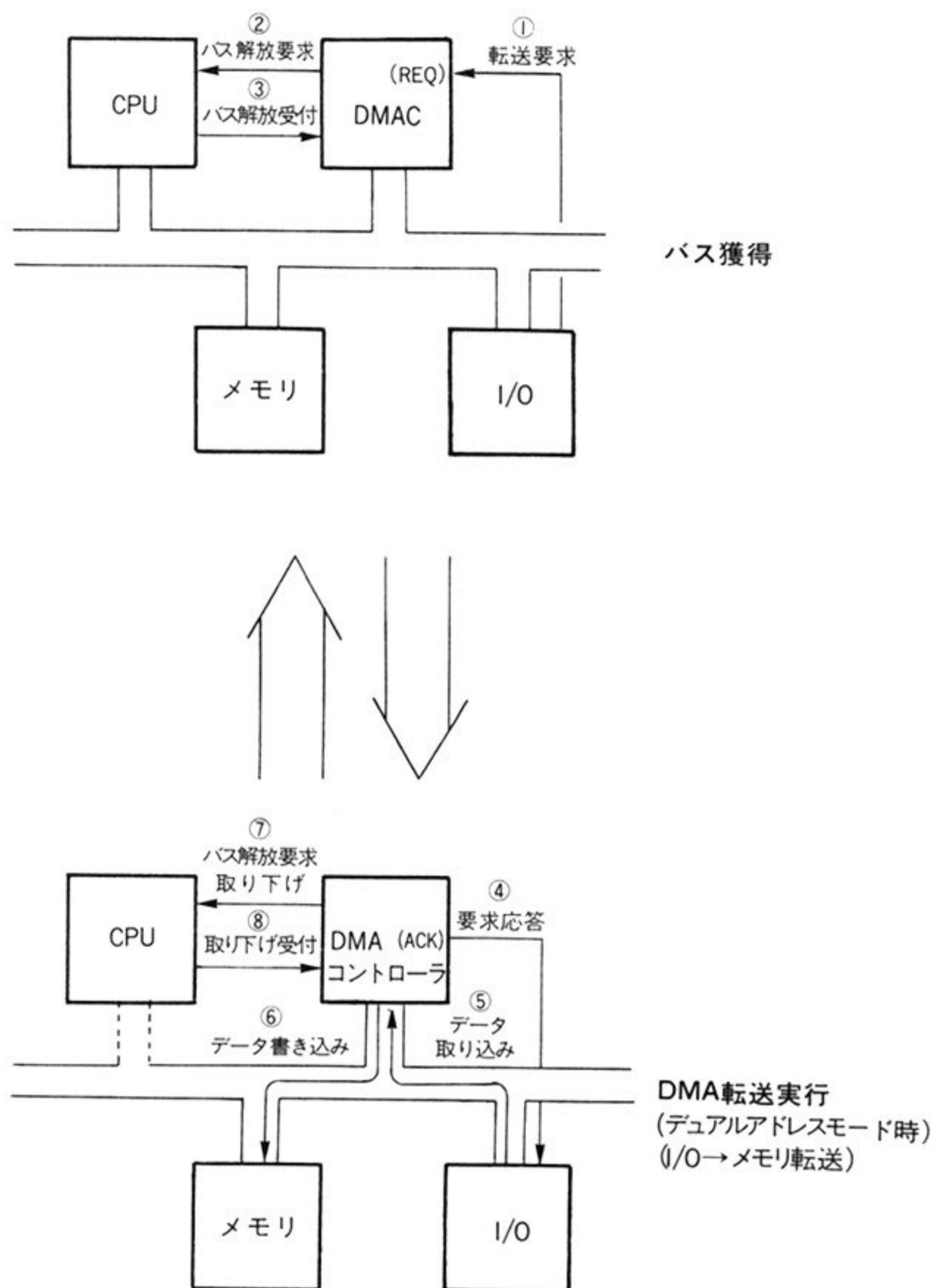
DMA

CPUを介さずにデータ転送を行うDMAは，割り込みでは応答するのが難しい高速なデータ転送や，CPUの処理動作とは独立したデータ転送処理をサポートします。ここでは，DMAの取り扱いについて説明します。

1 概要

DMAC(ダイレクトメモリアクセスコントローラ)は，メモリやI/Oのデータ転送を，CPUになりかわって行うICです。CPUを介さずに直接（ダイレクトに）データ転送を行うことから，このような名前がついています。通常，CPUには外部からの要求信号によって現在実行中の動作をきりのよいところで中断し，制御していた線（バス）をすべて電気的に切り離し，要求したデバイスにバスを解放する機能があります。もちろん，要求を取り下げれば，CPUは中断していた動作を再開します。DMACは，この機能を利用してデータの転送をCPUのプログラム実行に影響を与えずに行います(もちろん，データの転送元，転送先，転送する量などは，あらかじめDMACに設定しておく必要があります)。

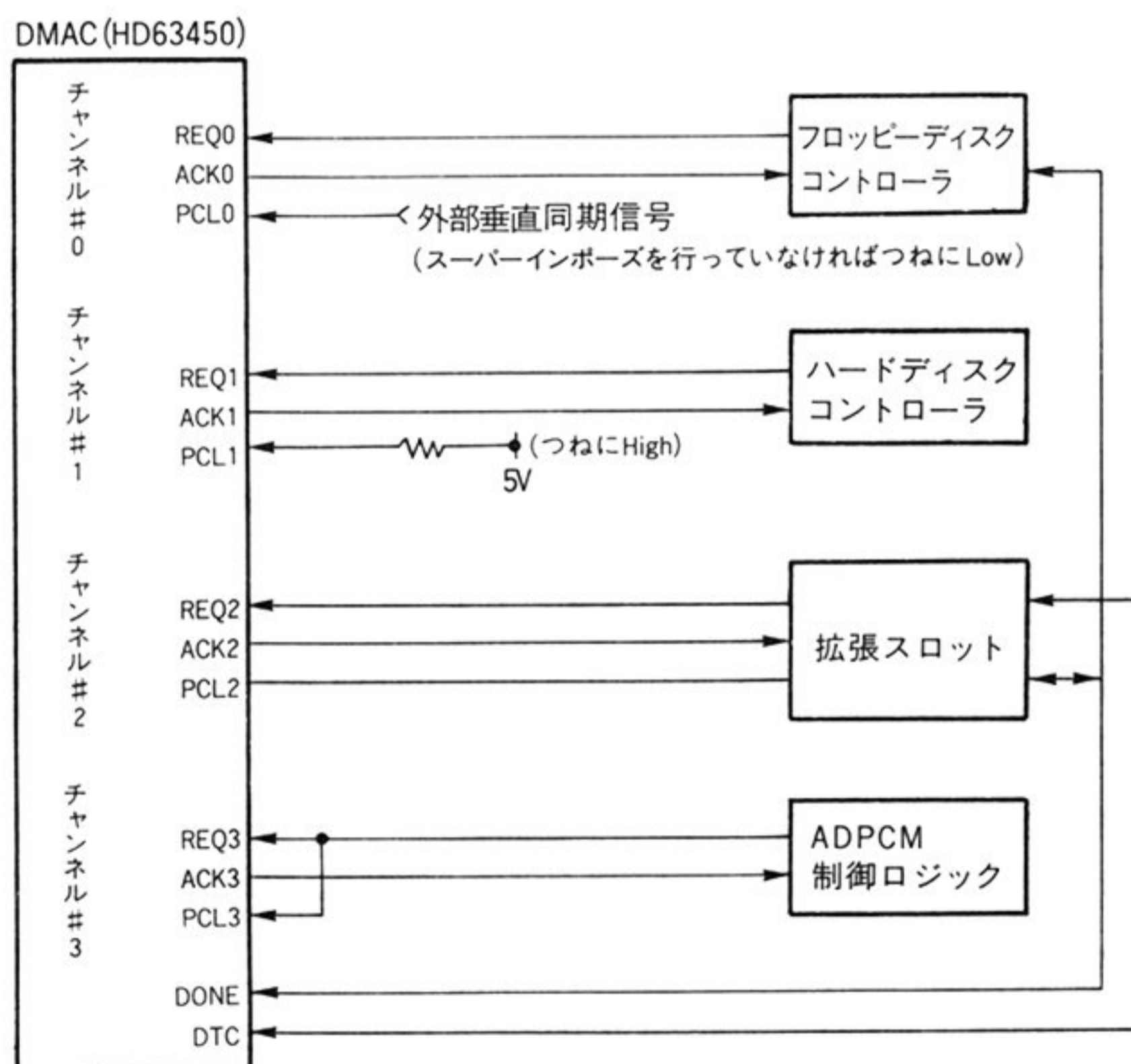
26ページの図1にDMACによるI/Oからメモリへのデータ転送動作の例を示します。I/Oからデータの転送要求が発生すると，DMACはCPUにバスの解放要求を行い，データの転送を実行した後，バス解放要求を取り下げるという動作を行います。この動作は純粋にハードウェア的に行われ，CPUがバスを取られる分だけ，プログラムの実行速度が落ちる以外はソフトウェアの動作には何の影響も与えません。

●図……1　DMACの動作概要

DMACの動作は，アプリケーションが気づかないところでデータ転送が行われるという点だけを見ると，割り込みによるデータ転送と似ていますが，ソフトウェアによる転送ではCPUがどのデバイスからの割り込みであるかの判定やレジスタの待避や復帰などの処理をする時間がかかるのに対して，DMAによる転送では要求が発生した時点でCPUがバスを使っていても，そのサイクルが終了したところで，すぐに転送が開始されるため，要求発生から実際の転送が開始されるまでの時間はDMAのほうが圧倒的に短く，高速のデータ転送が可能です。

X 68000では，高速なデータ転送を要求されるFD，HD，ADPCMにDMAを利用しています。

●2 DMACのチャンネル割り付け

図2にX 68000のDMACのチャンネルの割り付けを示します。X 68000で採用されたDMAC (HD 63450) は4つのチャンネルを持っており，このうちチャンネル# 0，# 1，# 3の3つがそれぞれFD，HD，ADPCMに割り付けられています。残るチャンネル# 2は使用されておらず，REQ (DMA転送要求信号)，ACK (応答信号)，PCL (汎用入力信号) などは拡張スロットに配線されています。このチャンネルはメモリ－メモリ間転送や拡張ボードで利用することができます。

●図……2　X 68000のDMACチャネル割り付け

＊全チャンネルともデュアルアドレスモードで使用する
＊チャンネル#0，#1，#3は，外部転送要求、サイクルスチールモードに設定すること
＊チャンネル#2はユーザ開放(メモリーメモリ転送にも利用可)

3 DMACのレジスター覧

図3にX 68000に採用されたDMAC, HD 63450の持つレジスター覧を示します。

各チャンネルごとに17個(GCRはDMAC全体に関係する設定を行うものなので，チャンネル#3用の空間である$E 840 FFだけにあります)のレジスタがあります。これらのレジスタのうち，CERはリードオンリー(読み出しのみ)ですが，それ以外のレジスタはすべてリード/ライトとも可能となっています。

これらのレジスタのうち，転送元や転送先のアドレス指定に使用されるのがMARとDAR，転送オペランド数を指定するのがMTCです。メモリーI/O間の転送を行う場合にはメモリアドレスをMARで，I/OアドレスをDARで指定します。

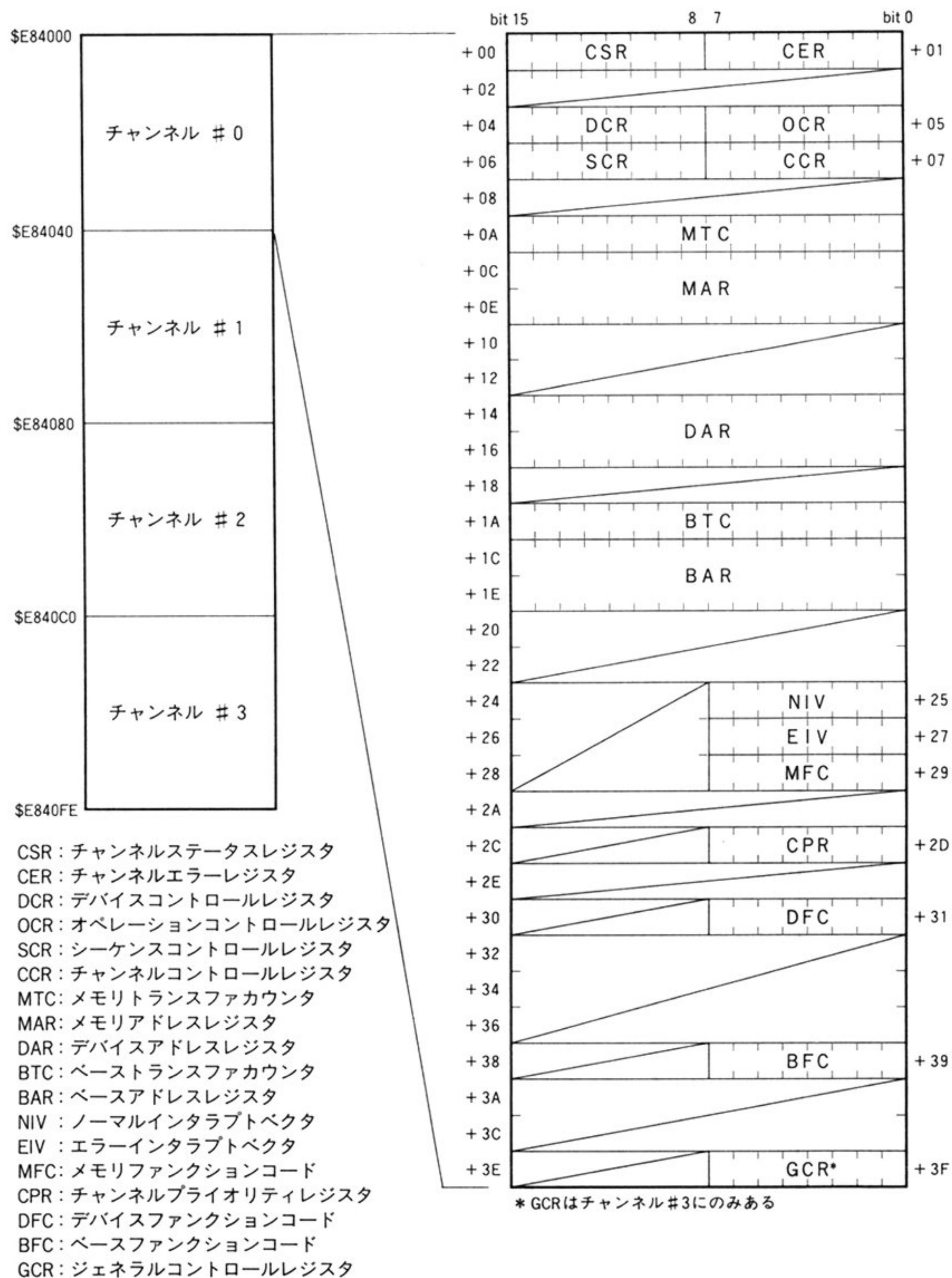
また，BARとBTCは複数ブロックの転送機能を利用するときに使用されます。その他のレジスタについては後で説明します。

4 DMACの動作モード

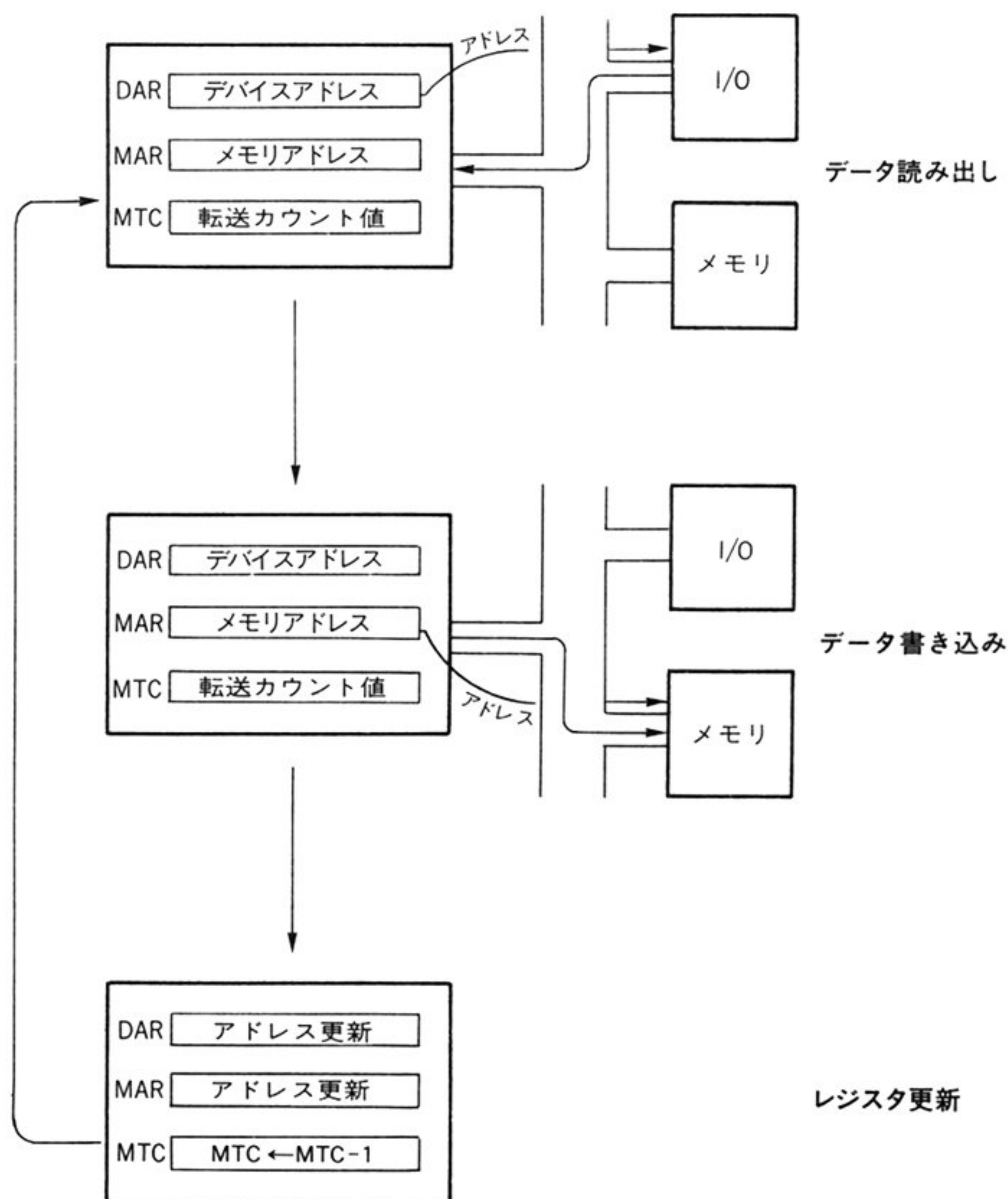
HD 63450は多くの動作モードを持っています。1オペランド(転送元から転送先への1回分)のデータの流し方で2通り，転送要求の発生方法や一度バスを持ったら一気に転送するか，1回ごとにCPUにバスを返すかといった1ブロック分の転送のモードで8通り，不連続なアドレスへの転送をサポートする複数ブロックの転送機能で3通りの動作モードがあります。

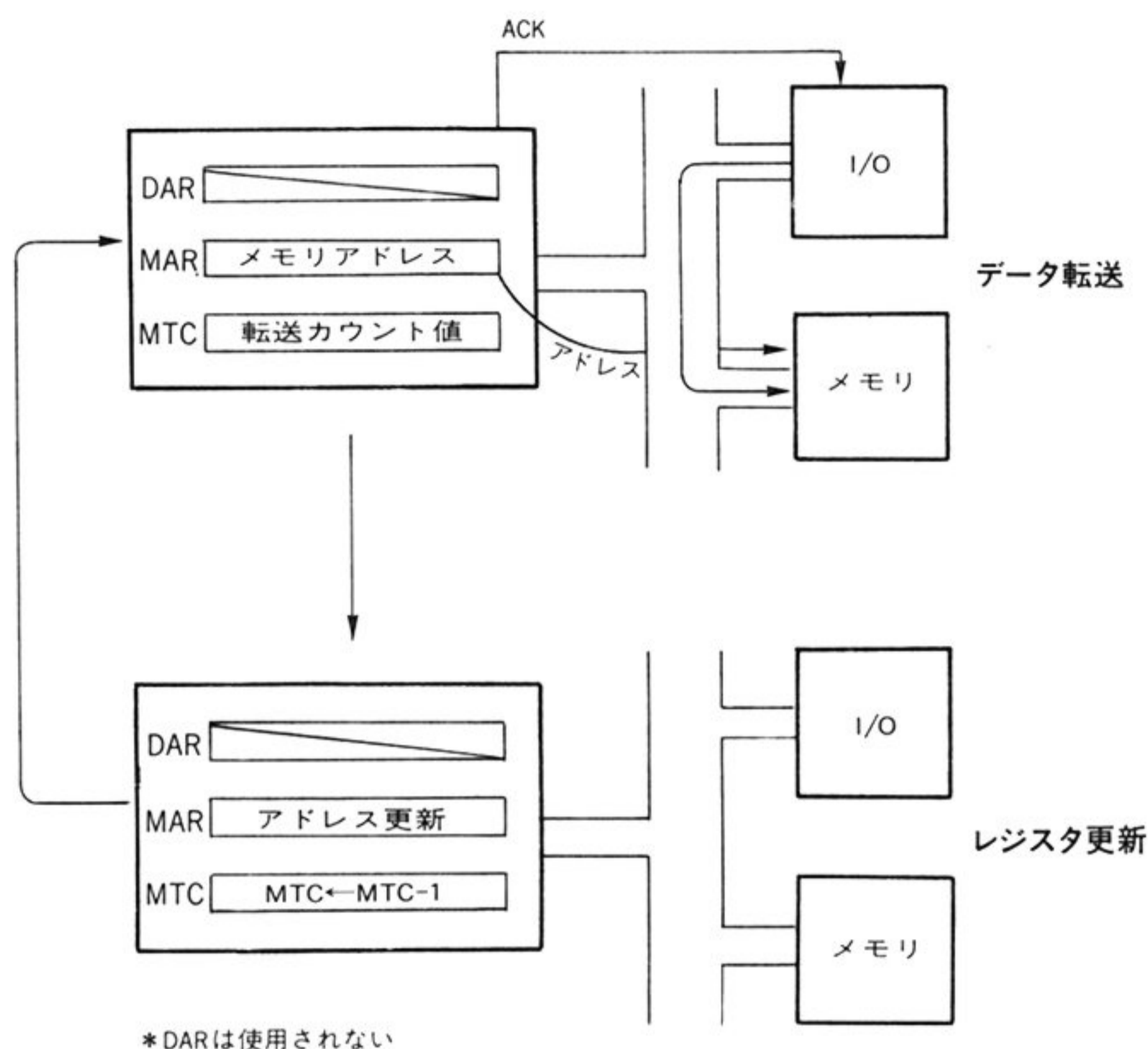
X 68000では#0, 1, 3の各チャンネルは用途が決まっており，設定内容も一部は固定となっていますが，チャンネル#2はさまざまな動作モードが選べるようになっていますので，ここでも一通りすべての動作モードを説明しておくことにします。

なお，DMACによる転送はメモリ→I/O，I/O→メモリ，メモリ→メモリ，I/O→I/Oの4通りが考えられますが，話をかんたんにするため，ここではI/O→メモリの転送動作で説明することにします。

●図……3　DMACレジスタ一覧

❹·1 1オペランド分の転送モード

DMACの動作を転送バス上のデータの流れで見ると，データをいったんDMAC内部に取り込み，次にDMACから書き込み動作を行うデュアルアドレスモードと，DMACはメモリアドレスを発生し，データは直接I/Oからメモリに流してしまうシングルアドレスモードに分類できます。それぞれの動作を図4と図5に示します。

●図……4　デュアルアドレスモード（I/O↔メモリ転送）（メモリ↔メモリ）

●図……5　シングルアドレスモード（I/O↔メモリ）

＊DARは使用されない

デュアルアドレスモードの場合，DMACはCPUによるアクセスと同じようにアドレスを与えてI/Oからデータを読み取り，メモリへ書き込みを行います。I/Oのアドレスとメモリのアドレス，転送回数はそれぞれDAR（デバイスアドレスレジスタ），MAR（メモリアドレスレジスタ），MTC（メモリトランスファカウンタ）で指定します。I/Oアドレスとメモリアドレスの2つのアドレスを用いるため，この動作をデュアルアドレスモードと呼んでいます。1回の転送が終わった後，DAR，MARを変更（増加/減少）するように設定されていれば，自動的に内容の更新が行われます。MTCは，1回の転送が終わるたびに1ずつ減らされていき，0になると転送動作は終了します。

デュアルアドレスモードは，周辺のハードウェアから見れば，CPUによるアクセスとなんら変わりませんので，I/Oとメモリの間だけではなく，メモリ－メモリ間やI/O－I/O間の転送も可能です。

シングルアドレスモードの場合，DMACはメモリアドレスしか発生せず，I/Oに対してはACK信号でデータ出力を要請します。I/Oとメモリは同じデータバスにつながっていますから，このデータはそのままメモリにも届きます。データをI/Oが，アドレスと書き込み制御をDMACが分担することで，1回のバス動作でデータ転送が終了するため，デュアルアドレスモ

ードよりも高速のデータ転送が可能です。

ただ，シングルアドレスモードは，I/O が 8 ビット幅でメモリが 16 ビット幅というように，ビット幅が異なるときには，DMA 転送のときだけ，偶数番地はデータバスの上位 8 ビットを，奇数番地は下位 8 ビットを利用するような細工が必要になるため，ハードウェアがやや複雑になります(デュアルアドレスモードのときには，このような処理は DMAC が行ってくれます)。また，シングルアドレスモードでは，その原理上，メモリーメモリ転送は行えません。

X 68000 では，各チャンネルともデュアルアドレスモードを利用するようになっています。

❹·2 1ブロック分の転送モード

HD 63450 の転送モードを図 6 に示します。

1 ブロック分の転送モードは，要求の発生源に注目すると，転送要求が毎回外部から与えられる外部要求転送モード，DMAC 内部で自動的に転送要求を発生するオートリクエストモード，最初のオペランドだけはオートリクエストで，以後は外部要求転送で動作するモードの 3 種類に分類できます。さらに，これらをバスの使い方で分類することで計 8 種類の転送モードに分類されます。

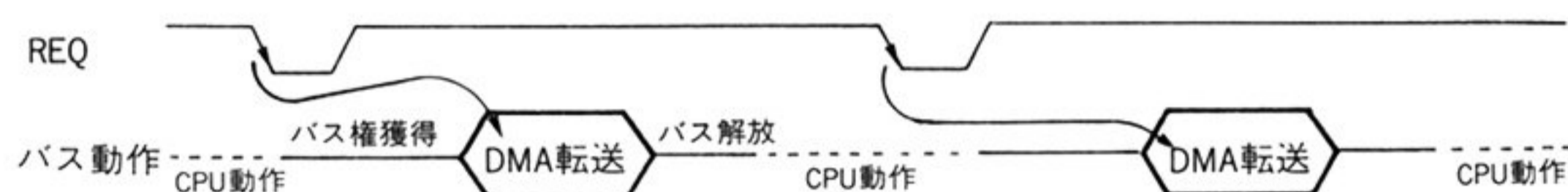
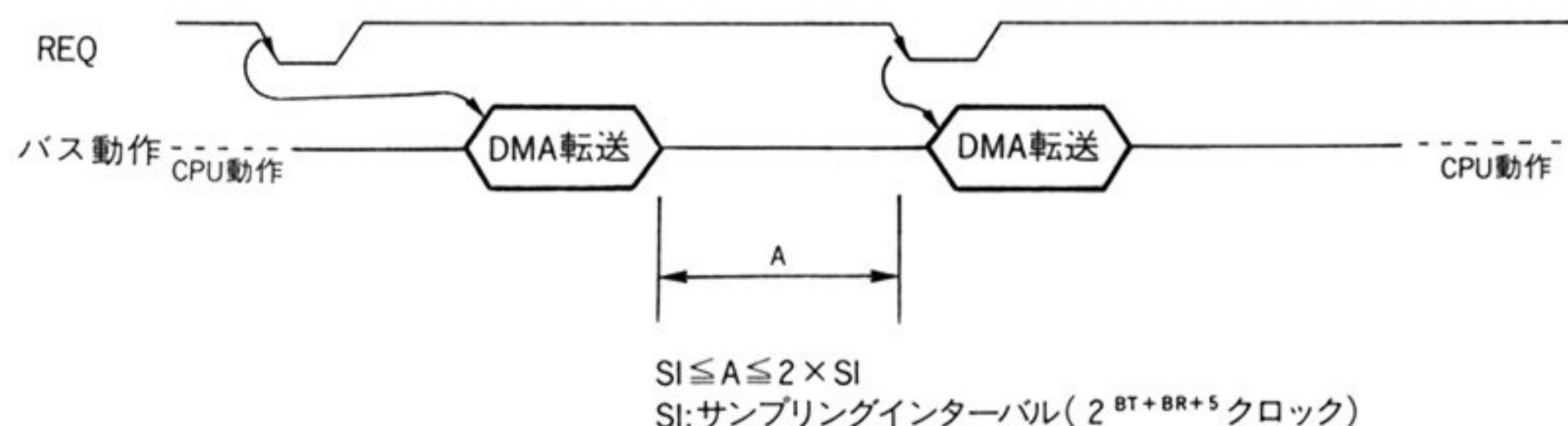
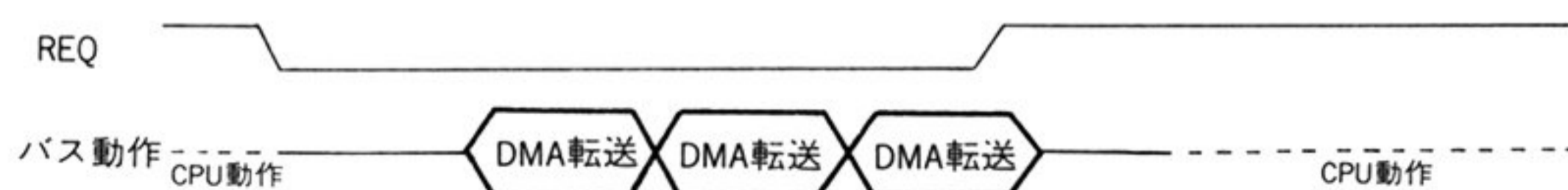
●図……6　1ブロック分の転送モード

転送モード		動作の概要
外部要求転送	ホールドなしサイクルスチールモード	要求をエッジで検出する 転送後，次の要求がなければバスを放す
	ホールド付きサイクルスチールモード	要求をエッジで検出する 転送後一定期間バスを持ったまま次の要求を待つ
	バーストモード	要求をレベルで検出する REQ がLow になっている間連続して転送する
オートリクエスト	最大速度	バスを持ったまま，最後まで転送する
	限定速度	GCRで規定される比率で定期的にバスを解放する
オートリクエスト＋外部転送要求		転送スタート後，1語目の転送はオートリクエスト。2語目以降は外部要求転送

❹・❷1 外部要求転送モード

外部要求転送モードは，さらにホールドなしサイクルスチールモード，ホールド付きサイクルスチールモード，バーストモードに分類されます。それぞれの動作タイミングの概略を図7に示します。

ホールドなしサイクルスチールモードはもっとも一般的な転送モードで，X 68000でも，FD，HD，ADPCMのどれも，このモードで利用します。REQ（転送要求）信号を立ち下がりエッジ（信号のHighからLowへの変化）でとらえ，転送終了時に次の要求が発生していなければすぐにCPUにバスを返します。

●図……7　外部転送要求モードによるDMA動作

（A）ホールドなしサイクルスチールモード

（B）ホールド付きサイクルスチールモード

（C）バーストモード

ホールド付きの場合には，転送が終了してもすぐにはCPUにバスを戻さず，次の要求がこないかどうか，しばらく様子を見ます。様子を見ている間に次の要求がくれば，ホールドなしの場合のようにふたたびCPUとバスの交換をする手間がかからない分だけ効率がよくなりますが，こない場合にはただよけいな時間がかかるだけになってしまいます。

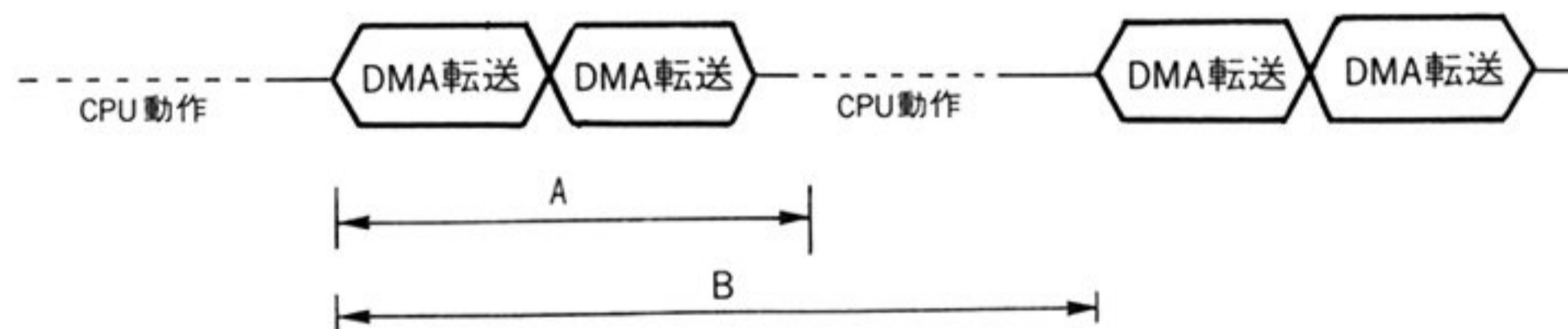
バーストモードは，REQ信号をレベルで判定し，REQ信号がLowになっている間，連続して転送を行います。要求が発生したら，転送サイズ分だけ一気に取り込むような用途に適したモードです。

❹・❷ 2 オートリクエストモード

オートリクエストモードでは，DMAC内部のレジスタの転送スタートビットをCPUが'1'にすることで転送が開始されます。転送要求を自分自身で発生させるため，このモードをオート（自動）リクエストモードと呼んでいます。オートリクエストモードには最大速度と限定速度の2種類の動作モードがあります。それぞれの動作タイミングの概略を図8に示します。

最大速度の場合には，いったん転送が開始されると，転送が終了するまでCPUにバスを返しません。データ転送速度は速くなりますが，大量のデータを最大速度で転送すると，長時間CPUが動けなくなってしまうという問題があります。

これと対照的なのが限定速度モードです。限定速度の場合にはDMACは定期的にバスを

●図……8　オートリクエストモードによるDMA動作

CPU に返し，バスの使用率があらかじめ設定された値になるように調整しながら動作します。限定速度での転送速度は当然最大速度よりは劣りますが，CPU が動作しながら転送動作が行えるため，システム全体としては都合のよいことも多くあります。

❹·3 複数ブロックの転送モード

通常，DMA 転送は，コントローラに設定した分(1 ブロック分)の転送を実行すると動作を終了し，CPU が次の設定を行うまで動作を停止したままになっています。複数ブロックの転送が必要な場合には，CPU が DMAC からの割り込みや動作ステータスによって転送終了を検出し，新しい転送アドレスなどを設定する手間がかかります。DMA 転送はこの間止まってしまいますから，複数ブロックへの転送が発生することがあらかじめわかっているときには，これはまったく無駄な時間になります。X 68000 の DMAC，HD63450 は，このような問題に対応して複数のブロックを連続して転送する機能をサポートしています。ただし，この機能では次々に設定できるのは MAR と MTC だけで，DAR は初期設定のまま全転送が終了するまで変更できません(インクリメント/デクリメントが指定されていれば，全転送が終了するまでインクリメント/デクリメントしつづけます)。

HD63450 の複数ブロック転送機能は，継続動作，アレイチェイン，リンクアレイチェインの 3 種類があります。次に，これらの機能を見ていきましょう。

❹·❸1 継続動作モード

継続動作モードは，DMAC が転送を実行している間に，次のメモリアドレス，転送カウンタ，ファンクションコードを BAR，BTC，BFC レジスタに設定する方法です。DMAC は，1 ブロック分の転送が終了すると，書き込まれた内容を MAR，MTC，MFC に取り込み，すぐに次の転送を開始します。この時点で CPU は，次のアドレスやカウント値を設定することができるようになります。

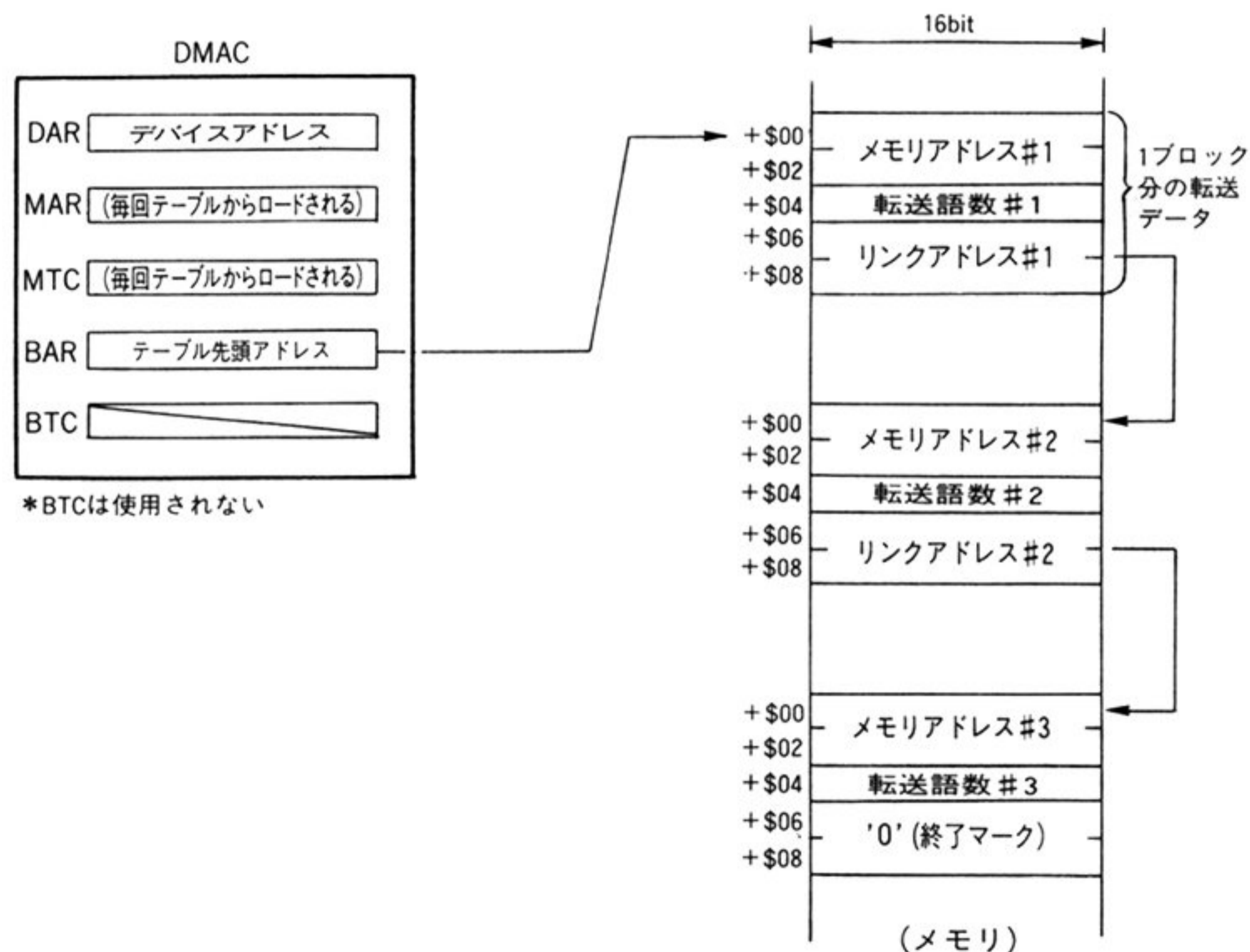
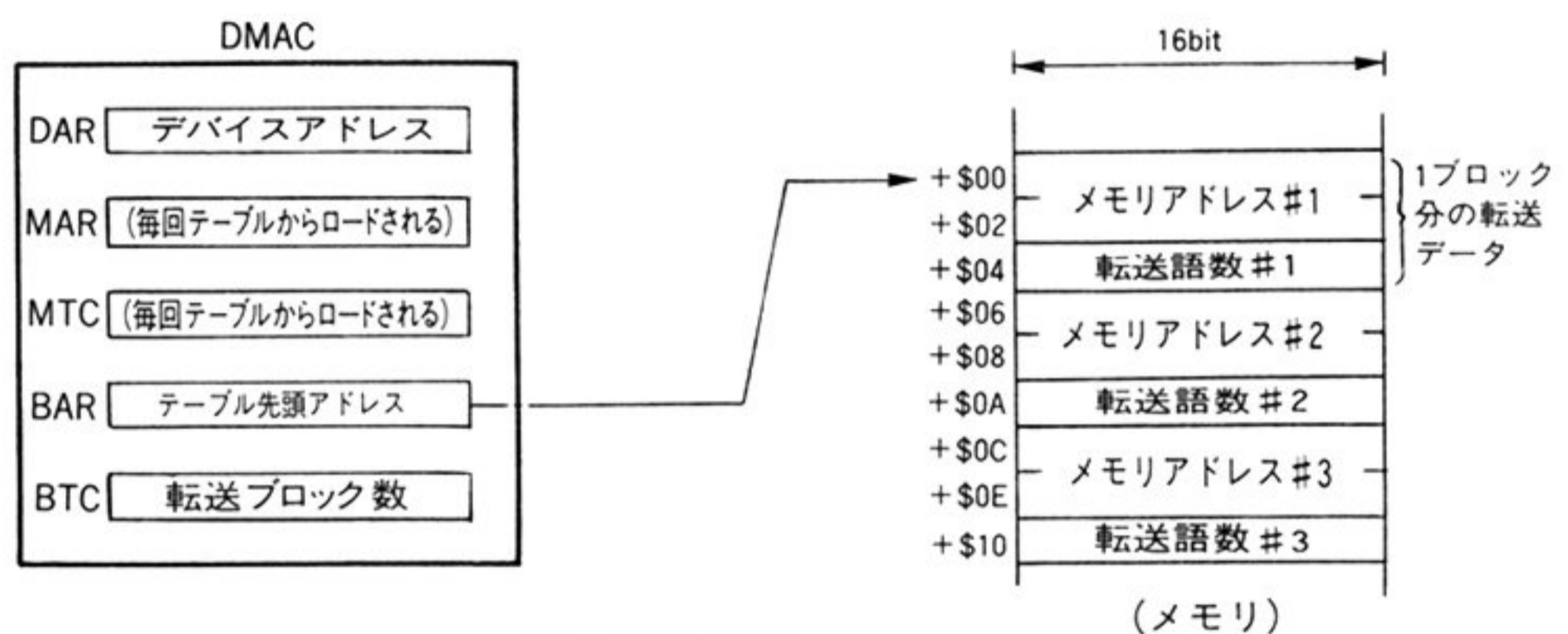
❹·❸2 アレイチェインモード，リンクアレイチェイン モード

アレイチェインモードとリンクアレイチェインモードは，メモリアドレスと転送カウンタのデータを示すテーブル（転送情報テーブル）をメモリ上に用意しておき，この先頭アドレスを

BAR に設定しておくと，DMAC 自体が次々に読み取って複数ブロックの転送を行うモードです。継続動作モードでは，あくまでも1ブロック分の転送終了ごとに CPU による再設定が必要なのに対し，アレイチェインモードとリンクアレイチェインモードの両モードは，よりインテリジェントな動作モードであるといえます。

アレイチェインとリンクアレイチェインの大きな違いは，転送ブロック情報テーブルの構造と動作終了条件にあります。各モードの転送情報テーブルの構造を図9に示します。

●図……9 アレイチェインとリンクアレイチェインモードの転送情報テーブル

アレイチェイン

リンクアレイチェイン

アレイチェインモードでは，各転送情報が連続したアドレスに配置され，DMACのBTCで転送するブロックの数（転送情報テーブルの数）を指定します。1ブロック分の転送が終了するたびにBTCの値は減らされていき，0になると動作終了となります。

リンクアレイチェインモードでは，1ブロック分の転送情報テーブルの後に次の転送情報テーブルのアドレス(リンクアドレス)が書き込まれています。DMACは，このリンクアドレスをたどって次の転送情報を得るわけです。リンクアドレスが0になっていると，転送を終了します。このため，リンクアレイチェインモードではBTCは使用されません。リンクアレイチェインモードは，アレイチェインモードのように転送情報テーブルを連続したアドレスに配置する必要がなく，自由度が高いモードであるといえます。

アレイチェインモードとリンクアレイチェインモードの違いを図10にまとめておきましたので参考にしてください。

●図……10　アレイチェインモードとリンクアレイチェインモードの比較

転送モード	アレイチェインモード	リンクアレイチェインモード
BARの内容	転送情報テーブルの先頭アドレス	同　左
BTCの内容	転送ブロックの数	〔使用されない〕
転送情報テーブルの内容	転送アドレス 転送語数	転送アドレス 転送語数 リンクアドレス
転送終了条件	BTC＝0	リンクアドレス＝0

●5 DMACのレジスタの内容

DMAC，HD 63450は多くのレジスタを持っていますが，設定そのものはそれほどむずかしいものではありません。ここではDMACの持つ各レジスタの内容について説明し，具体的な設定方法について解説していくことにしましょう。

なお，説明やレジスタのビット配置は，X 68000のdb.xで見るときに都合のよいように，別々のレジスタであっても，1ワード単位で読み出すことができるものについてはワード単位

で扱っています。これらのレジスタは読み出しを行うときはワード単位でもかまいませんが，書き込みはワード単位で行えないものもあります。たとえば，CCR レジスタの STR ビットなどは，ワードアクセスで'1'をセットしようとすると，動作タイミングエラーになってしまいます。とくに意味のないかぎり，各レジスタごとにアクセスするようにしたほうがよいでしょう。

❺·1 CSR，CER

CSR（チャンネルステータスレジスタ）と CER（チャンネルエラーレジスタ）のビット配置を図 11 に示します。

CSR はチャンネルの動作状態や PCL ラインステータスを示すもので，CER はなんらかのエラーが発生したときにエラー内容の詳細を示すために使用されます。

CSR のうち ACT と PCS 以外のビットは，いったん '1' になると，そのビットを '1' にしたデータを書き込むか，リセットがかかるまで'1'のままになります。とくに COC，BTC，NDT，ERR，ACT ビットが'1'になっているときには次の転送動作を行うことができません(動作タイミングエラーになる)ので，使用前にチェックしてクリアするようにしてください。

❺·❶1 COC（チャンネルオペレーションコンプリート）

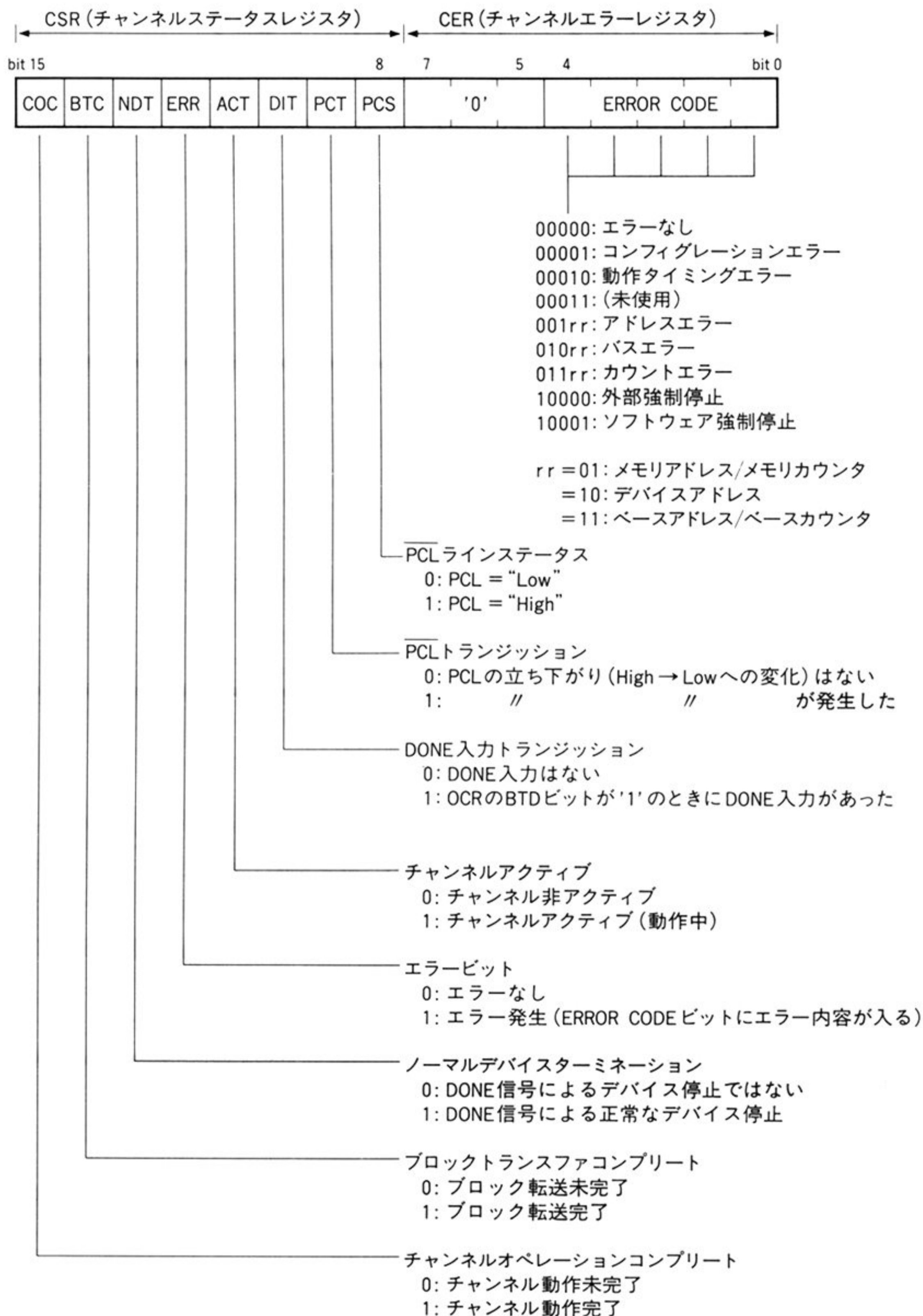
COC ビットはチャンネルの動作が終了したときに'1'になります。再度，そのチャンネルを使用するときには COC ビットをクリア（'0'にする）しておかなければなりません。クリアせずに次の転送を開始しようとすると，動作タイミングエラーになります。

❺·❶2 BTC（ブロックトランスファコンプリート）

BTC（ブロックトランスファカウンタ）レジスタと名称が同じなので，混同しないように気をつけてください。本書では，たんに BTC とした場合には BTC レジスタを指し，CSR の BTC ビットの場合には 'BTC ビット' と表記することにします。

BTC ビットは継続動作を行っているとき（CCR の CNT ビットを '1' にしているとき）に MTC が 0 になるとセットされます。つまり，継続動作モードのときに 1 ブロック分のデータが転送し終わったことを示すのが BTC ビットというわけです。

BTC ビットも，再度 CNT ビットを '1' にして継続動作を再開させる前にクリアしておかな

●図……11 CSR：チャンネルステータスレジスタ CER：チャンネルエラーレジスタ（＋$00）

いと，動作タイミングエラーになります。

❺・❶ 3 NDT（ノーマルデバイスターミネーション）

HD 63450には，DMA要求を行ったI/Oデバイスが全データの転送を終了したことを示すために，DONE信号ピンが用意されています。NDTビットは，この信号によってDMA転送が終了したことを示す信号です。

NDTビットも，再度DMA転送を行う前にクリアしておかないと，動作タイミングエラーになります。

❺・❶ 4 ERR（エラー）

なんらかのエラーが発生すると'1'になります。このとき，エラーの内容がCERレジスタにセットされています。

ERRビットも，次の転送を行う前にクリアしておかないと，動作タイミングエラーになります。

❺・❶ 5 ACT（チャンネルアクティブ）

ACTビットはチャンネルが動作中であることを示すビットです。CCRのSTR(スタート)ビットがセットされ，転送動作が開始すると'1'になり，転送が終了すると'0'になります。COCビットと似ていますが，COCビットがたんにチャンネル動作の終了を表すだけであり，ソフトウェアで'0'にクリアされるのに対し，ACTビットはチャンネルの動作中だけ'1'になるという点が異なります。

❺・❶ 6 DIT（DONE入力トランジッション）

DONE付きの複数ブロック転送モードが選択されたとき（OCRのBTDビットを'1'に設定する)，DONE入力によるブロックの転送の中断が起こると'1'になります。

❺·❶7 PCT (PCLトランジッション)

HD 63450には各チャンネルごとに汎用の入出力ラインとしてPCLピンが用意されています(このピンの機能はDCRのPCLビットやDCRのDTYPビットで決められます)。PCTビットは，この信号ピンがどのようにプログラムされているかに関係なく，HighからLowへの変化があると'1'にセットされます。

X 68000では，チャンネル#0のPCLに外部ビデオ信号の垂直同期信号が，チャンネル#3のPCLにはADPCMのDMA要求信号が接続されています。

❺·❶8 PCS (PCLラインステータス)

PCSビットはPCLピンの状態がそのまま読み出されます。PCLピンがどのようにプログラムされているかには関係ありません。PCLピンがHighなら'1'，Lowならば'0'になります。

❺·❶9 ERROR CODE

CSRのERRビットがセットされたとき，CERにはエラーの内容を示すデータが入ります。それぞれのエラーステータスと，発生する要因を次に示します。

1)コンフィグレーションエラー

- チェインモード時にCNT (継続動作指示) ビットがセットされたとき
- シングルアドレスモード (DCRのDTYPビットで指定) 時にデバイスポートサイズ (DCRのDPSビットで指定) とオペランドサイズ (OCRのSIZEビットで指定) が一致していない場合
- デュアルアドレスモードで外部転送要求 (OCRのREQGビット='10'または'11') のとき，デバイスポートサイズを16ビット，オペランドサイズを8ビットに設定したとき
- DCR，OCR，SCRの各ビットに未定義の値をセットした場合
- デュアルアドレスモードでデバイスポートサイズが8ビットのとき以外に，OCRのSIZEビットに'11'を設定した場合

2)動作タイミングエラー

- ・チェインモードでSTRビット（CCRレジスタ）とACTビット（CSRレジスタ）の両方ともセットされていないときにCNTビットをセットした場合
- ・CSR中のCOC，BTC，NDT，ERR，ACTのいずれかのビットが'1'になっているときにSTRビットをセットした場合
- ・STRビットかACTビットが'1'になっている（チャンネルが動作を開始している）ときにDCR，OCR，SCR，CCR，MAR，DAR，MTC，MFC，DFCのいずれかに書き込みを行った場合
- ・BTCビットとACTビットが'1'になっているときにCNTビットをセットした場合

3)アドレスエラー

- ・ワードやロングワードオペランドの転送を奇数番地から行おうとした場合（実際にアクセスが行われた時点でエラーが発生する）
- ・DMAバスサイクルのときにDMAのCSピンやIACKピンをLowにした場合（X 68000ではハードウェアの故障でもないかぎり，このようなことは起こりません）

4)バスエラー

- ・DMAがバスを使用しているときにバスエラーが発生した場合

5)カウントエラー

- ・チェインモード以外のときにMTCレジスタに0を設定し，STRビットをセットしたとき（0バイトの転送を行おうとしたとき）
- ・アレイチェインモードモードでBTCに0を設定したまま，STRビットをセットした場合
- ・チェインモード，コンティニューモードのときにメモリ（チェインモード時）やBTC（継続動作モード時）からMTCに0がロードされたとき

6)強制終了

- ・PCLがアボート入力信号としてプログラムされており，STRビットかACTビットが'1'になっているときにアボート信号を与えた場合

7)ソフトウェアアボート

- ・STRビットかACTビットが'1'になっているときにCCRレジスタのSAB（ソフトウェアアボート）ビットがセットされたとき

❺·2 DCR, OCR

DCR（デバイスコントロールレジスタ）とOCR（オペレーションコントロールレジスタ）のビット配置を44ページの図12に示します。DCRは，DMACに接続されるI/Oデバイスの種別やPCLピンの機能を設定するために，OCRはDMACの転送モードを設定するために使用されます。

❺·❷1 XRM（エクスターナルリクエストモード）

XRMは外部要求転送のときの転送モードを設定するのに使用します。この設定が有効になるのは，OCRレジスタのREQGビットが'10'か'11'になっているときです。Human 68 Kでは，チャンネル#0，1，3とも'10'（ホールドなしサイクルスチールモード）で使用しています

❺·❷2 DTYP（デバイスタイプ）

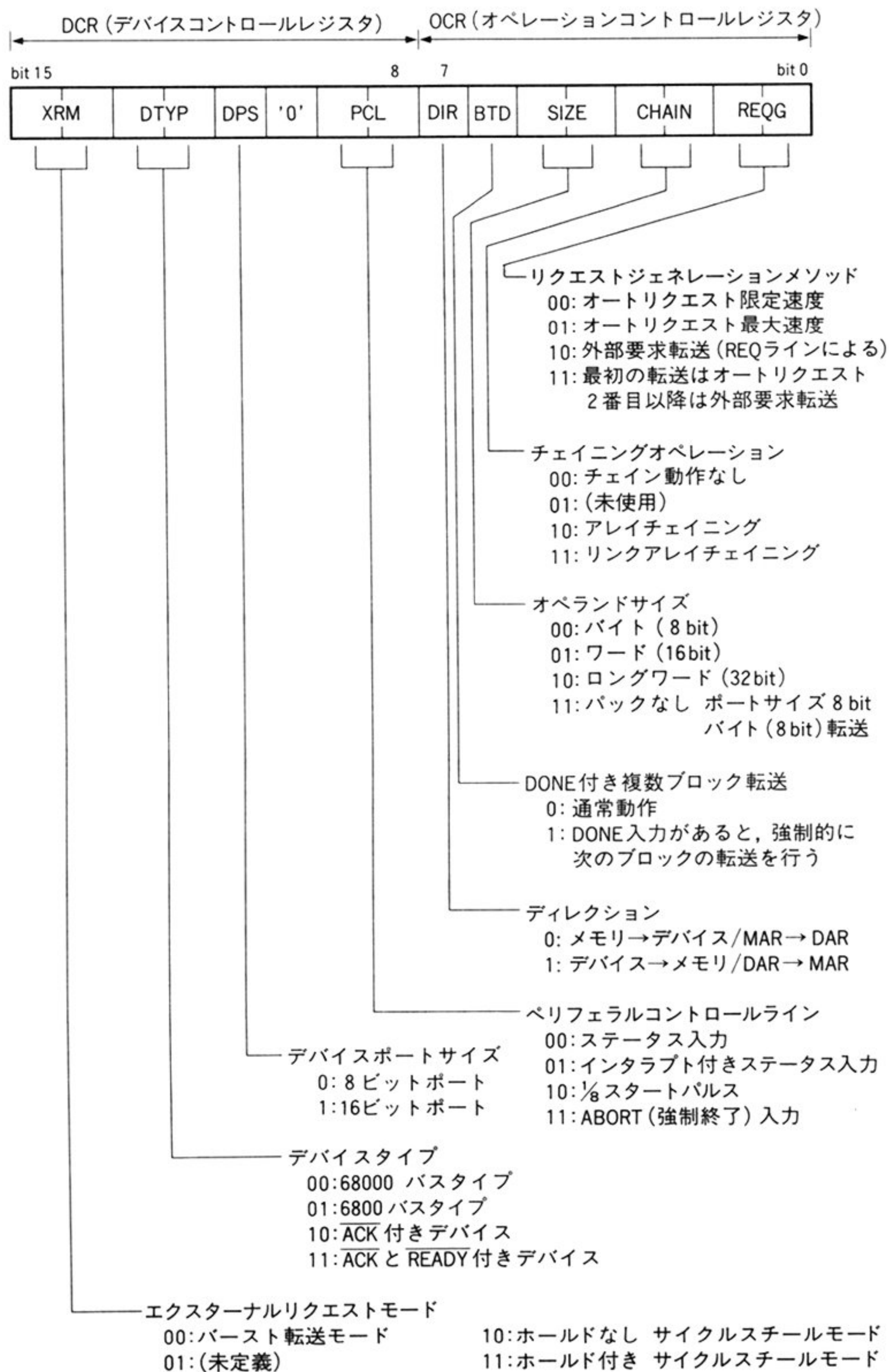
DMACに接続されているI/Oのアクセス方法を設定します。設定値'00'および'01'はデュアルアドレスモード，'10'と'11'はシングルアドレスモードの動作になります。

'00'は68000の信号をそのまま使ったようなデバイスで，CPUでもリード/ライトできるようなものに適用します。メモリはこのタイプに分類されます。X 68000の場合，内部I/Oもすべてこのタイプですから，通常は'00'以外を設定することはありません。

'01'の6800タイプというのは，モトローラの8ビットCPUである6800用の周辺デバイスをつないだときに設定するモードです。このとき，DMACのPCLラインは6800タイプのデバイスが動作タイミングを取るためのEクロックの入力端子として動作するようになります（PCLビットによる設定は無視されます）。6800ファミリーはかなり古いデバイスということもあり，X 68000用の拡張ボードなどで使用されることもないと思われます。

'10'と'11'はともにシングルアドレスモードです。X 68000では基本的にシングルアドレスモードのサポートはうたっていませんから，拡張ボードで使われることもまずないと思われます。したがって，以下の説明は読み飛ばしてもかまいません。

'10'と'11'の違いは，I/OデバイスとDMACとの間の転送タイミングの取り方にあります。'10'のときにはDMACからI/Oに対してACK信号を返すことでI/O側はデータの入

●図……12　DCR：デバイスコントロールレジスタ　OCR：オペレーションコントロールレジスタ（＋$04）

出力を行います。'11'はI/O側の応答が遅く，DMAが出力してくるACK信号のタイミングでは間にあわない場合に使用されるモードです。I/OからDMACに対してデータの入出力準備ができるまで待ってもらう信号（READY信号）を出力することで，DMACにウェイトをかけるわけです。DMAC側では，PCLラインがこのREADY信号の入力ピンとなります。'11'に設定したとき，PCLビットによる設定は無視されます。'10'と'11'のどちらに設定するかはハードウェアの作り方で決まります。

❺·❷3 DPS（デバイスポートサイズ）

接続されているI/Oが8ビットポートであるか，16ビットポートであるかを決めるビットです。デュアルアドレスモードのときは，DAR（デバイスアドレスレジスタ）でアクセスされる側のデバイスが8ビットアクセスしかできないのか，16ビットアクセスもできるのかを設定することになります。DPSが0なら8ビットポート，1ならば16ビットポートであることを示します。メモリは16ビットポートの扱いになります。

X68000の場合，FD，HD，ADPCMはすべて8ビットポートです。チャンネル#2を使ったメモリーメモリ間転送は通常16ビットポートに設定して行いますが，256色や16色モードのときのグラフィック画面のように，上位ビットが意味を持たないようなときには8ビットポートに設定して転送を行うことができます。

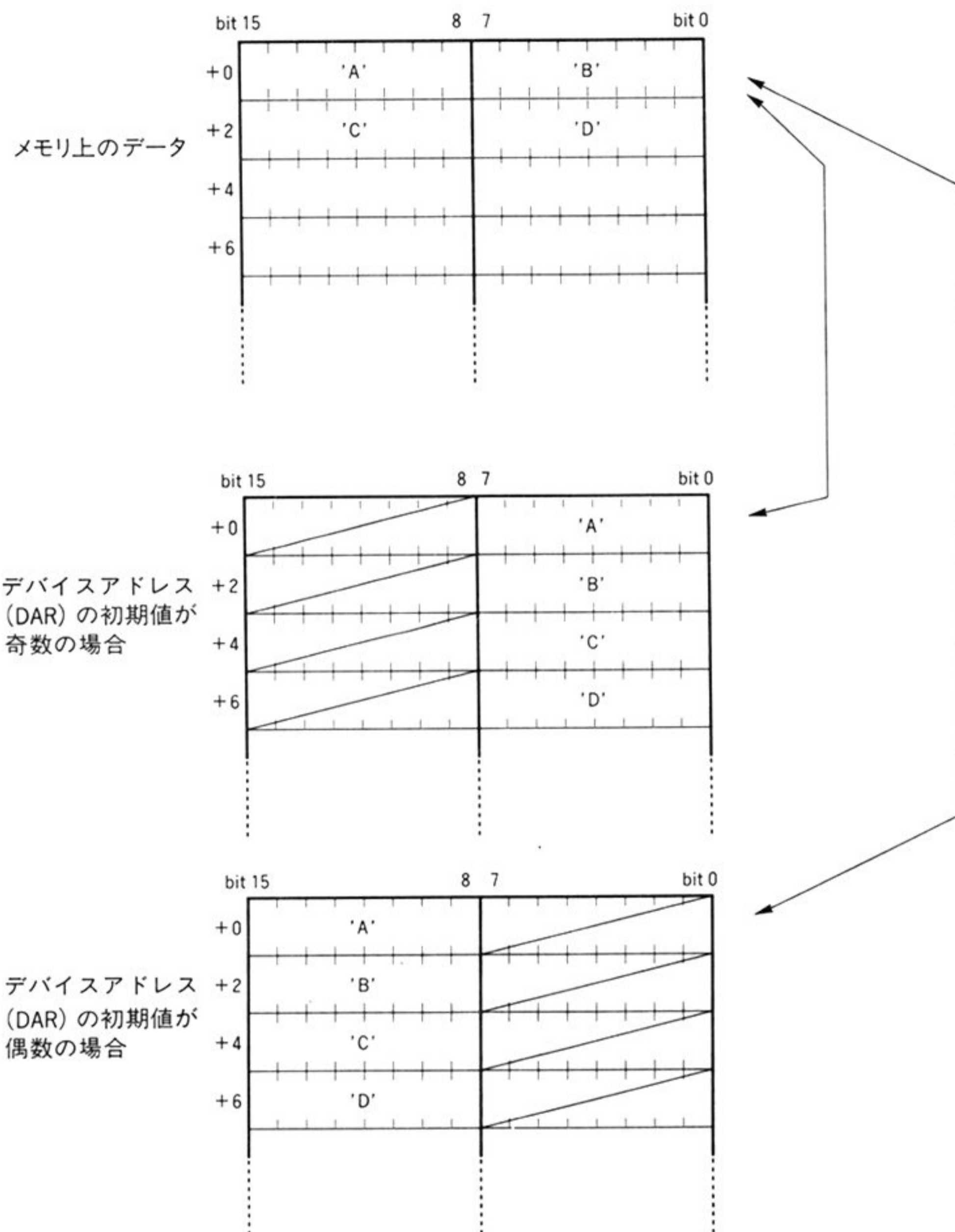
DPSが'0'（8ビットポート）に設定されており，DARが変化するように設定しているときには転送先の番地が2番地おきになることに気をつけてください。46ページの図13にメモリからDPSを'0'に設定したデバイスへの転送がどのように行われるかを示します。

DARの初期値が偶数の場合には上位8ビット，奇数の場合には下位8ビットだけを飛び飛びにアクセスしていくような動作になります。X68000のグラフィック画面では16色や256色モードのときも1ドットは1ワードであり，上位ビットを無視するようになっているため，DPSを'0'にしてDMA転送する方法が使えます。

❺·❷4 PCL（ペリフェラルコントロールライン）

DCRのDTYPビットが'01'（6800バスタイプ）以外のときに，DMACのPCLピンの機能を設定します。

'00'や'01'に設定したとき，PCLピンはステータス入力ピンとなります。'01'に設定したときはPCLピンの立ち下がり（HighからLowへの変化）で割り込みが発生します。PCLピ

●図……13 デバイスポートサイズが 8 ビットのときの転送

ンによる割り込みであることは，割り込み処理ルーチンの中で CSR を読むと，PCT ビットが立っていることから判断することができます。

'10' に設定すると，PCL ピンはチャンネルがアクティブになったことを外部に示す出力信号として動作します。PCL ピンは通常 High レベルですが，チャンネルがアクティブになった後，4 クロックサイクルの間だけ Low となります。

'11' に設定されると，PCL ピンは DMA 転送の強制終了（ABORT）入力信号ピンとして動作するようになります。この信号によって DMA 転送が終了するとエラー扱いとなり，CSR

の ERR ビットが'1'になり，CER には$10（外部強制停止）がセットされます。

❺·❷ 5 DIR（ディレクション）

DMA によるデータの転送方向を設定します。このビットを '0' にするとメモリから I/O への転送，'1' にすると I/O からメモリへの転送を行います。デュアルアドレスモードのときには，'0'にすると MAR（メモリアドレスレジスタ）で示される番地から DAR（デバイスアドレスレジスタ）で示される番地への転送，'1' にすると逆方向への転送になります。

❺·❷ 6 BTD（DONE付き複数ブロック転送）

HD 63450 には，複数ブロックの転送時に DONE 入力を使ってそのブロックの転送を中断し，強制的に次のブロックの転送に移る，DONE 付き複数ブロック転送の機能があります。このビットが '1' になっていると，このモードが選択されます。

❺·❷ 7 SIZE（オペランドサイズ）

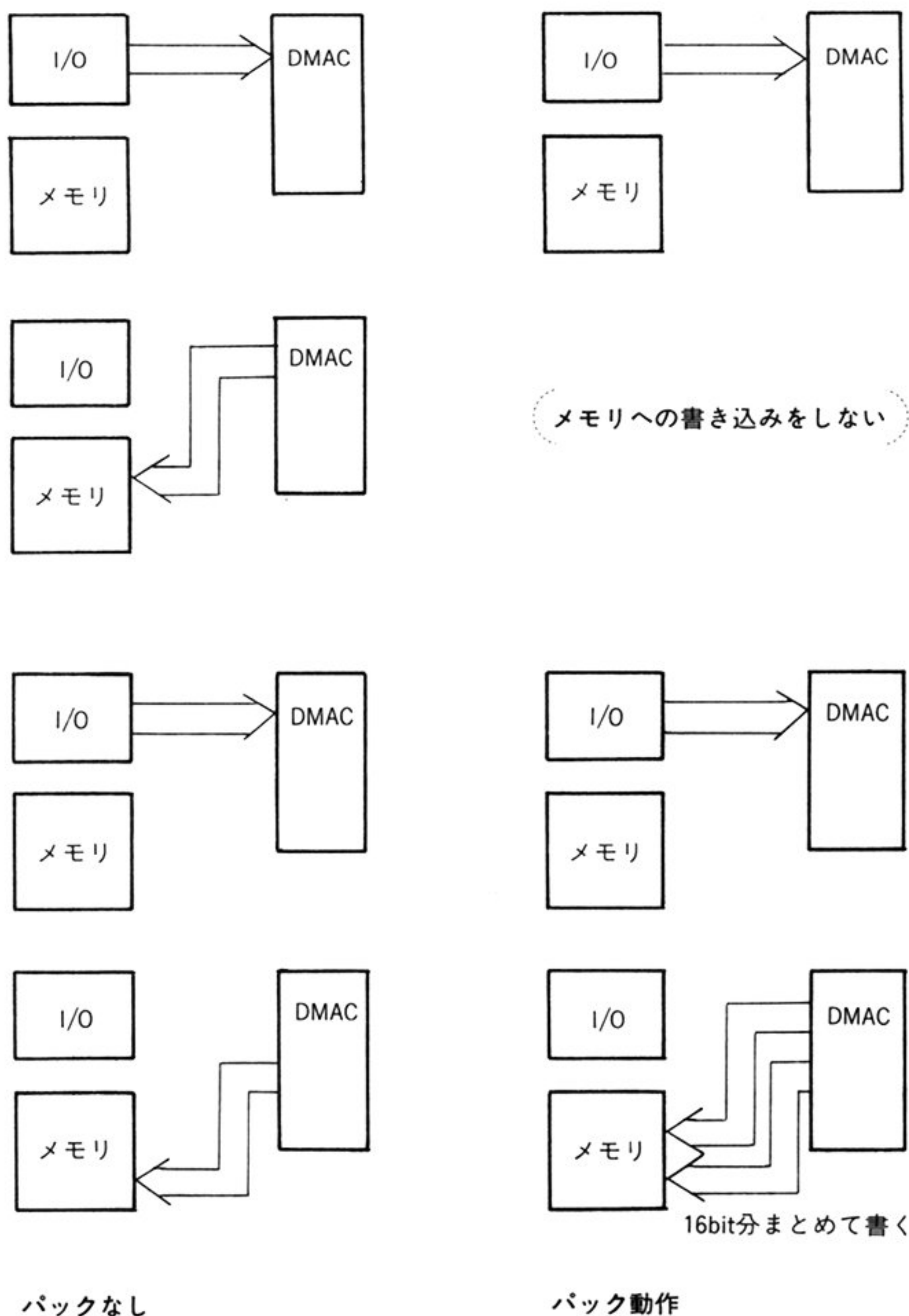
データの転送を行う単位を，バイト（8 ビット），ワード（16 ビット），ロングワード（32 ビット）のいずれにするかを設定するビットです。ただし，オペランドサイズが 8 ビットのときには，DMAC はバスの使用効率を上げるため，可能なかぎりデータをまとめて転送するパック動作を行いますので，バス上の実際の動作が SIZE の指定どおりになっていないことがあります。

たとえば，SIZE が 8 ビット，DPS が 8 ビットに設定されており，転送バイト数が 2 バイト以上あり，次にアクセスするメモリ番地が偶数という場合を考えてみます。

この場合，メモリへはワード単位でアクセスできるため，DMAC は I/O アクセスを 2 回，メモリアクセスを 1 回という転送サイクルを実行します。48 ページの図 14 にパック動作が行われないときと行われた場合の I/O からメモリへのデータ転送の例を示します。パック動作が行われると，I/O から 2 回読み取った後でメモリへ書き込むようにすることでメモリへのアクセスを 1 回分節約するわけです。逆方向（メモリから I/O）への転送ならば，メモリからワード単位で読み取った後，I/O へのバイトアクセスを 1 回行い，次の I/O アクセスは先ほど読み出しておいたデータを I/O に転送することで，メモリアクセスを 1 回節約するわけです。

ワードアクセスする側（先ほどの例ではメモリ）のアドレスが奇数であった場合や，ワードアクセスする側のアドレスが変化しないように設定されている（SCR の MAC や DAC で設定する）場合にはパック動作は行われず，SIZE の設定どおりバイト単位で転送が行われます。

SIZE ビットの '11' の設定は，先ほどの例のような 8 ビットポートとメモリの間のデータ転送時のパック動作を禁止し，必ずバイト単位で I/O とメモリを 1 回ずつアクセスするようにするものです。X 68000 の DOS，Human 68 K では，チャンネル# 0，# 1，# 3 とも SIZE ビッ

●図……14　DMAC　パック動作の例

トを '11' に設定して使っています。

❺・❷8 CHAIN(チェイニングオペレーション)

すでに述べたとおり，HD 63450 は複数ブロック転送をサポートする機能として，DMAC 自体がメモリ上の転送情報テーブルを読み取りながら動くアレイチェイニング動作や，リンクアレイチェイニング動作が行えるようになっています。

CHAIN ビットは，このチェイン動作を行わせるか，行わせるのであればアレイチェイン動作にするのか，リンクアレイチェインにするのかを決めるビットです。'00' のときにはチェイン動作は行われません。'10' のときはアレイチェイニング動作，'11' のときにはリンクアレイチェニング動作になります。

❺・❷9 REQG(リクエストジェネレーションメソッド)

1ブロック分の転送モードであるオートリクエスト，外部転送要求などの転送モードを選択するビットです。'00' と '01' はともにオートリクエストで，外部からの転送要求信号が発生しない，メモリーメモリ間転送などに使用されます。'01' のときには最大速度ですから，転送終了までバスを取ったままになりますが，'10' のときには GCR で設定された比率で間欠的に転送を行います。

'10' のときは外部の I/O からの REQ 信号に応答して転送を実行する外部要求転送モードに，'11' のときはチャンネルが動作開始して 1 回目の転送はオートリクエスト，それ以降は外部要求転送で動作します。

DTYP, DPS, SIZE, REQG ビットには設定できない組み合わせがあります。50 ページの図 15 に DMAC がサポートしているモードをまとめましたので参照してください。

●図……15　DMACがサポートするモード

アドレスモード (DTYP)	デバイスポートサイズ (DPS)	転送要求発生法 (REQG)	オペランドサイズ (SIZE)		
			バイト	ワード	ロングワード
デュアルアドレスモード (DTYP＝'00'or'01')	8 bit	'00', '01', '10', '11'	○	○	○
	16bit	'00', '01'	○	○	○
	16bit	'10', '11'	×	○	○
シングルアドレスモード (DTYP＝'10'or'11')	8 bit	'00', '01', '10', '11'	○	×	×
	16bit	'00', '01', '10', '11'	×	○	×

○：設定可
×：設定不可
DTYP, DPS：DCR(デバイスコントロールレジスタ)中のビット
REQG, SIZE：OCR(オペレーションコントロールレジスタ)中のビット

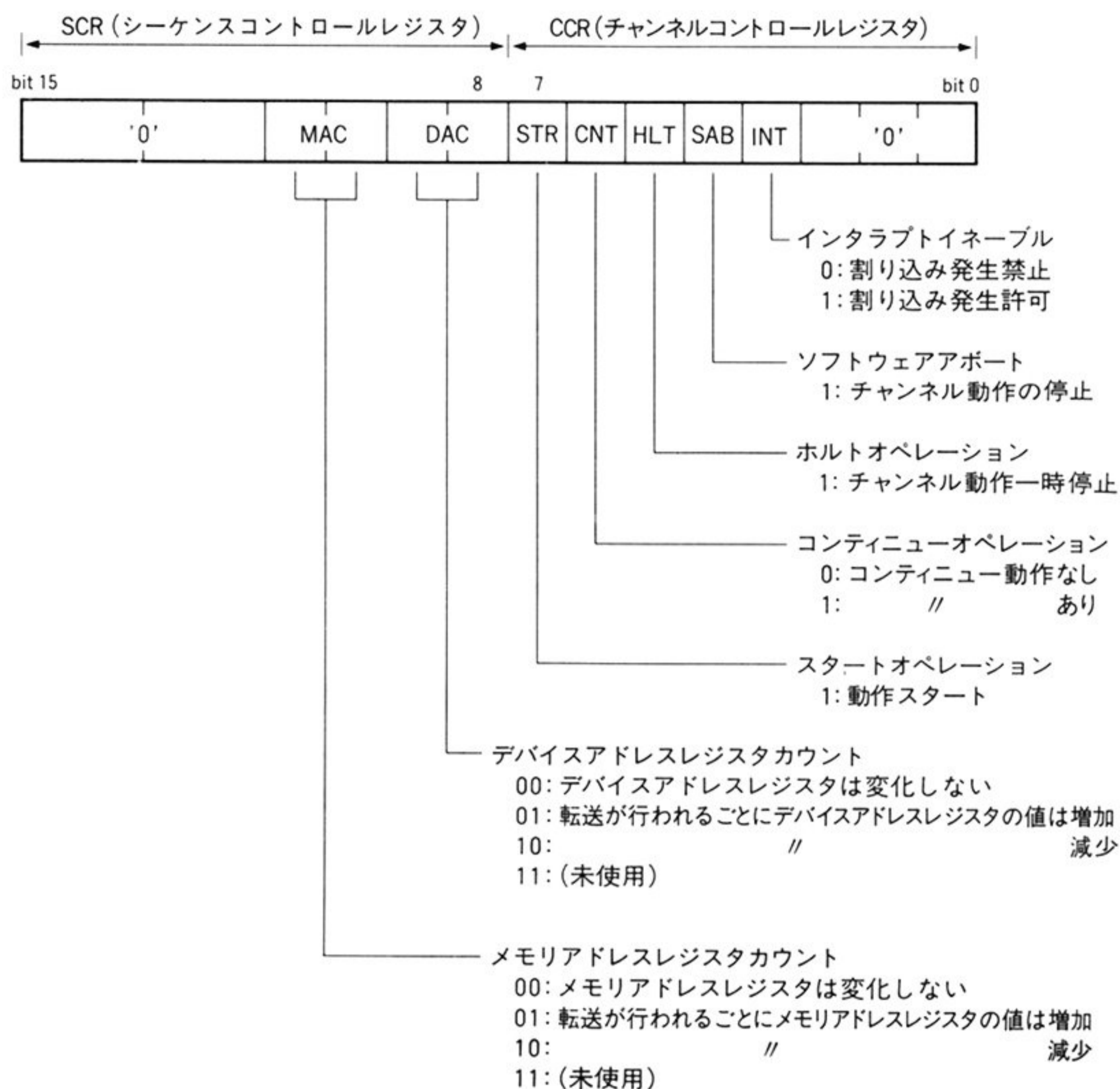
❺・3　SCR, CCR

SCR（シーケンスコントロールレジスタ）と，CCR（チャンネルコントロールレジスタ）のビット配置を図16に示します。SCRは転送元や転送先アドレスの増減の制御，CCRはチャンネルの動作の開始/停止や割り込みマスクなどを行うのに使用されます。

❺・❸1　MAC（メモリアドレスレジスタカウント）

MACは，DMA転送のたびにMAR（メモリアドレスレジスタ）の値を増減させるか否かを決定します。MACが'00'のときはMARは変化しません。'01'のときには転送が行われるたびに増加，'10'のときには減少します。

シングルアドレスモードのときには，この変化量はオペランドサイズに一致しますが，デュアルアドレスモードのときにはDCRのDPSビットやOCRのSIZEビットの設定によって変化します。52ページの図17にデュアルアドレスモードのときの1オペランドの転送ごとのデータ転送の形態やアドレスがどれだけ増減されるかなどをまとめてみました。オペランドサイズがバイト単位のときにはパック動作がからむため厄介なように思えますが，これはDMACが転送効率を上げるために陰でどのように動作するかということであって，CPU側が気にかける必要はほとんどありません（転送がエラーで終了したときの要因解析を行うときには知っておく必要があるでしょうが）。

●図……16　SCR：シーケンスコントロールレジスタ　CCR：チャンネルコントロールレジスタ（＋$06）

❺・❸ 2　DAC（デバイスアドレスレジスタカウント）

デュアルアドレスモードのときにデバイス側のアドレスを指定するDAR（デバイスアドレスレジスタ）の増減の指定を行うビットです。

❺・❸ 3　STR（スタートオペレーション）

DMA転送の開始を指示するビットです。通常は'0'で，このビットを'1'にするとDMA

●図……17　デュアルアドレスモードの動作

デバイスポートサイズ (DPS)	オペランドサイズ (SIZE)	メモリアクセス 〔サイズ〕×〔回数〕	デバイスアクセス 〔サイズ〕×〔回数〕	アドレス増減量		
				メモリアドレス	デバイスアドレス	
8 bit	バイト(SIZE='00')	ワード×1	バイト×2	±2	±4	*1
	ワード(SIZE='01')	ワード×1	バイト×2	±2	±4	
	ロングワード(SIZE='10')	ワード×2	バイト×4	±4	±8	
	パックなし バイト(SIZE='11')	バイト×1	バイト×1	±1	±2	
16bit	バイト(SIZE='00')	ワード×1	ワード×1	±2	±2	*2
	ワード(SIZE='01')	ワード×1	ワード×1	±2	±2	
	ロングワード(SIZE='10')	ワード×2	ワード×2	±4	±4	

＊1：パック動作が行われない場合，メモリ，デバイスともバイト×1
　　アドレス更新分は，メモリアドレスは＋1，デバイスアドレスは＋2
＊2：パック動作が行われない場合，メモリ，デバイスともバイト×1
　　アドレス更新分は，メモリアドレス，デバイスアドレスとも＋1

転送が開始されます。STR ビットに '0' を書き込んでも動作は停止しません。強制的に終了させたいときは SAB ビットを，一時停止させたいときは HLT ビットを '1' にします。

STR ビットを '1' にするときは，DAC レジスタへのアクセスはバイト単位で行ってください。ワードやロングワードでアクセスすると動作タイミングエラーになります。Human 68K の db. x の me（メモリエディット）コマンドはリード/ライトともワード単位で行われますので，DMAC にスタートをかけられません。注意してください。

❺・❸ 4 CNT（コンティニューオペレーション）

複数ブロック転送のうちの継続動作を行わせるときに使用するビットです。STR ビットか CSR の ACT ビットが '1' になっているとき（転送動作中のとき）に，次の転送アドレスや転送数，ファンクションコードを，それぞれ BAR, BTC, BFC の各レジスタにセットした後で CNT ビットを '1' にすると継続動作になります。

STR や ACT ビットが '1' になっていないときに CNT ビットを '1' にすると動作タイミングエラーになります。また，チェインモードが指定されているとき（OCR の CHAIN ビットが '10' や '11' のとき）に CNT ビットを '1' にすると，コンフィグレーションエラーになります。

❺·❸ 5 HLT（ホルトオペレーション）

転送動作中に HLT ビットを '1' にすると，一時的に転送動作を停止します。HLT ビットが '0' に戻ると，中断していた転送動作を再開します。外部要求転送のとき，HLT ビットによって動作を中断していても，次の要求が発生したかどうかのセンスは行っています。

ただし，バースト転送モードのときには HLT ビットが '0' に戻った後，最初の転送が開始されるまで，I/O デバイスは REQ 信号を出し続けなくてはなりません。

❺·❸ 6 SAB（ソフトウェアアボート）

'1' にすると転送動作を強制的に終了させます。このとき，CSR の ERR ビットが '1' になり，CER には$11（ソフトウェア強制停止）がセットされます。ERR ビットが '1' になったときに SAB ビットは自動的にクリアされるようになっていますので，SAB ビットはいつでも '0' が読み出されます。

❺·❸ 7 INT（インタラプトイネーブル）

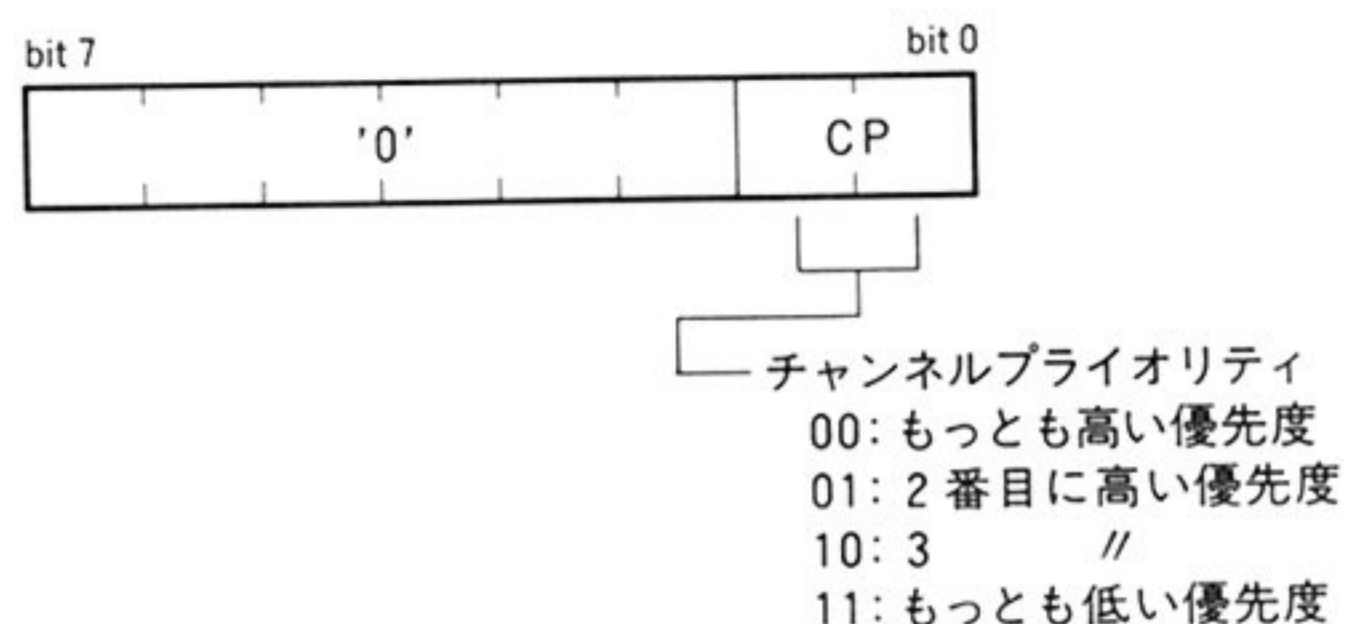
チャンネルの動作が終了したり，エラーが発生したときに CPU に対して割り込みをかけるか否かを指定します。'1' になっていると割り込み発生を行い，'0' になっていると割り込みを発生しなくなります。

割り込みの発生する条件は，INT が '1' で CSR レジスタの COC，BTC，ERR，NDT，PCT のいずれかが '1' になったときです。ただし，PCT は，DCR の PCL ビットで割り込み付きステータス入力にプログラムされているときだけ割り込み要因となります。

割り込み発生時の割り込みベクタ番号は NIV（ノーマルインタラプトベクタ）レジスタ，EIV（エラーインタラプトベクタ）レジスタで指定します。ERR ビットが '1' になっているときには EIV が，それ以外のときには NIV の値が使用されます。

❺·4 CPR

CPR（チャンネルプライオリティレジスタ）のビット配置を 54 ページの図 18 に示します。

●図……18　チャンネルプライオリティレジスタ（＋$2D）

CPRは，DMACの持つ4つのチャンネル間のプライオリティ（優先順位）を決定するものです。プライオリティは'00'がもっとも高く，'11'がもっとも低くなっています。複数のチャンネルから同時に要求があった場合，プライオリティの高いほうのチャンネルがサービスされます。

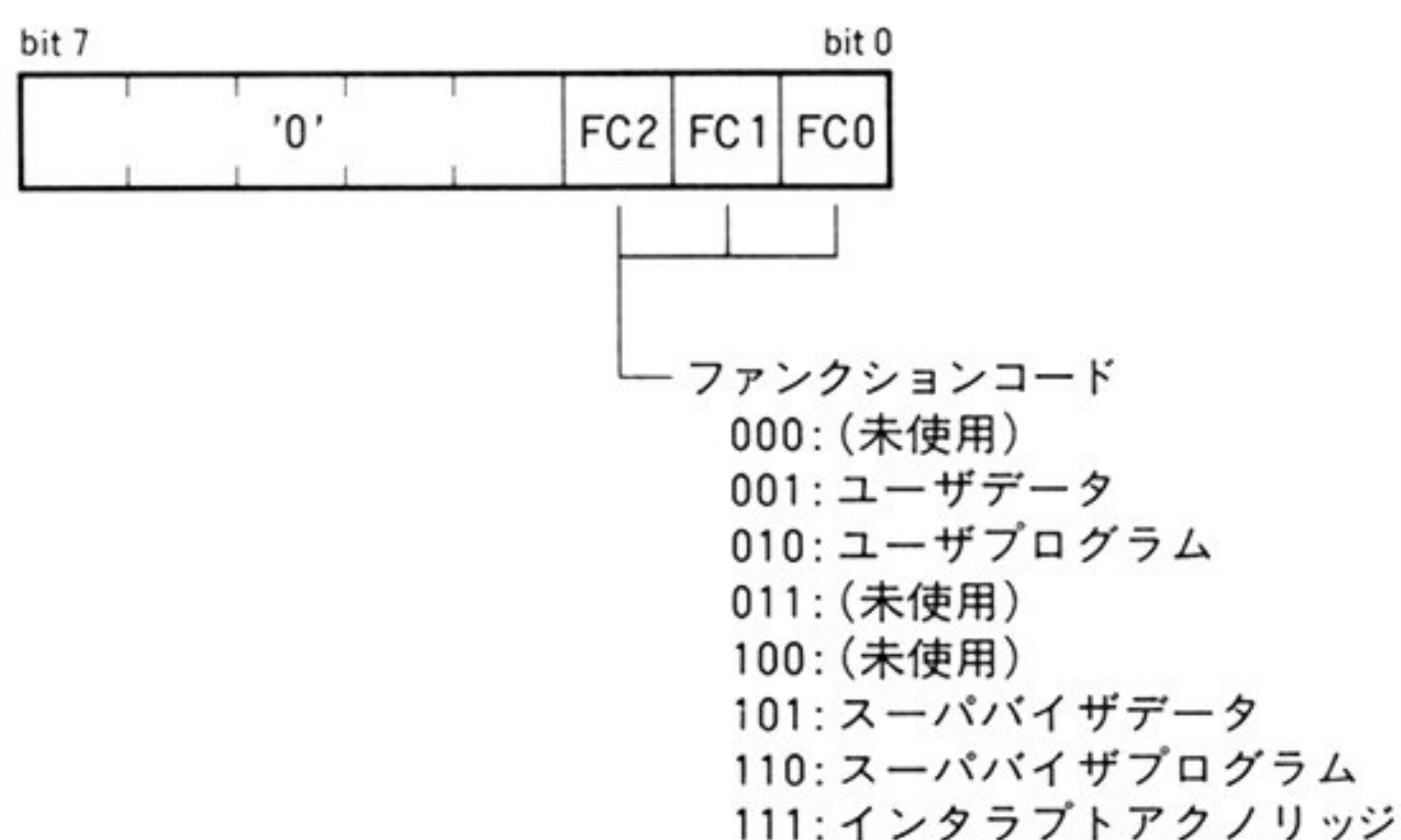
複数のチャンネルに同じプライオリティを設定することも可能です。この場合，同一プライオリティのものどうしの間ではサービスされたものがもっとも低いプライオリティとなり，巡回サービスされるラウンドロビン方式でサービスが行われます。

❺·5 MFC, DFC, BFC

MFC（メモリファンクションコード），DFC（デバイスファンクションコード），BFC（ベースファンクションコード）のビット配置を図19に示します。68000 CPUは，メモリやI/Oをアクセスするときにファンクションコードと呼ばれる3ビットのステータス信号を外部に出力します。このステータス信号は，今回のアクセスが，ユーザモードでのアクセスなのか，スーパーバイザモードでのアクセスなのか，またデータアクセスなのか，プログラムの読み出しなのか，あるいは割り込みへの応答サイクルなのかといった情報を示すのに使われます。

X 68000の場合，Human 68 Kの本体やワークエリアのある低い番地やVRAMやI/Oのある領域をユーザモードからアクセスしようとするとバスエラーが発生しますが，このプロテクション機構は，このファンクションコードを使って行っているのです。

DMACもCPUに準じ，ファンクションコードを出力できるようになっています。DMACが出力するアドレスを保持するレジスタはMAR（メモリアドレスレジスタ），DAR（デバイスアドレスレジスタ），BAR（ベースアドレスレジスタ）の3本がありますので，ファンクションコードも各レジスタごとに指定できるように3つ用意されています。MARでアクセスするときに使われるのがMFC，DARのときはDFC，BARにはBFCが使用されます。継続動作モードのときには，次に使用されるMFCをBFCに設定します。

●図……19　MFC/DFC/BFC:ファンクションコードレジスタ（＋$29/＋$31/＋$39）

X 68000で通常使うときにはファンクションコードは'101',すなわちスーパーバイザデータ(スーパーバイザ状態でのデータアクセス）にしておけばよいでしょう。

❺·6　GCR

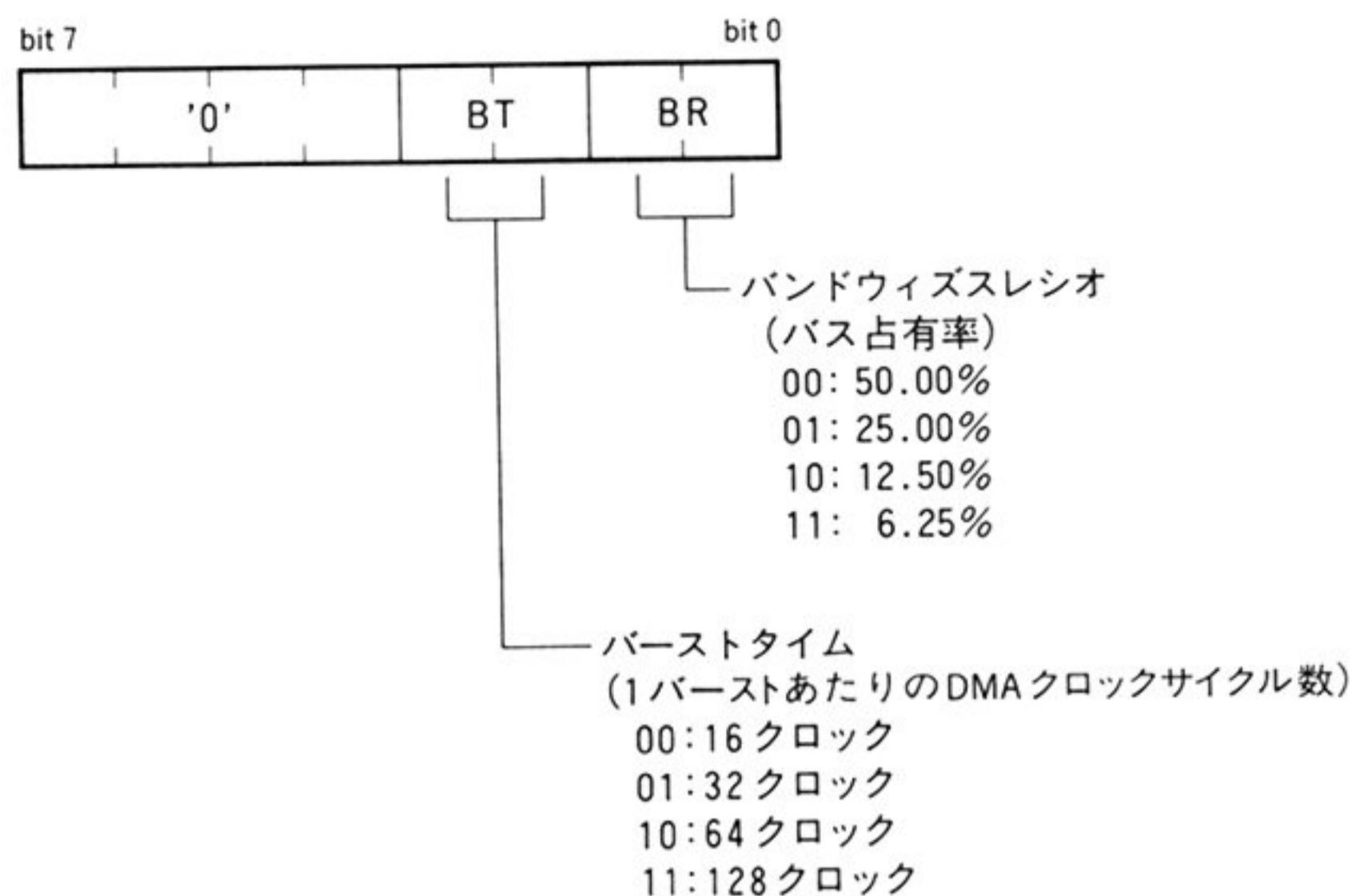
GCR（ジェネラルコントロールレジスタ）のビット配置を56ページの図20に示します。GCRは，限定速度で転送を行うときのバスの占有のしかたを制御します。

❺·❻1　BT（バーストタイム）

限定速度で動作するとき，限定速度でのDMA転送要求を発生する期間(オートリクエストインターバル）をクロック数で設定します。オートリクエストインターバルは2^{BT+4}クロックとなります。

❺·❻2　BR（バーストウィズスレシオ）

限定速度で動くときのバスの使用率を決定するビットです。DMACは，CPUが出力するBGACK(バス解放要求受付）信号を監視して，CPU以外のデバイス（X 68000ではDMACしかありませんが）がバスを使用している期間が全サイクルの$2^{-(BT+1)}$になるように限定速度

●図……20　ジェネラルコントロールレジスタ（＋$FF）

での転送を実行します。

いま，BTビットに'00'，BRビットに'01'を設定したとします。このとき，オートリクエストインターバルは16クロック，バス占有率は25パーセントとなります。また，DMACがバスの使用率のサンプリングを行う期間は$2^{BT+4+BR+1}$クロックです。この例ではサンプリング期間は64クロックとなります。

DMACは，64クロックの間，BGACK信号を監視し，CPU以外のデバイスがバスを使っている期間を測定します。もし，この期間が16クロック以下であれば，次の64クロックの期間が始まってから16クロックの間，限定速度によるDMA転送要求を発生します。もし，16クロック以上バスが使用されていれば，次の64クロックの間，限定速度によるDMA転送要求を発生しません。このような動作により，長い期間で見ると，CPU以外のデバイスによるバス占有率は25パーセント程度になります。

限定速度でのバス使用率が，転送するチャンネルの使用率ではなく，CPU以外の全デバイスが使っている期間で算出されることに注意してください。限定速度以外に設定されたチャンネルがあまり頻繁にDMA転送を行っていると，限定速度に設定したチャンネルはいつまでたっても転送が実行できないことになります。

●6 Human 68Kの初期設定値

Human 68 K による DMAC の設定値を図 21 に示しますので，DMAC を自分でイニシャライズして使用するときの参考にしてください。Human 68 K は，DMAC のイニシャライズを起動時に行うのではなく，それぞれのチャンネルを使用するときにはじめて行うようです。このため，フロッピーディスクから起動したままの状態でチャンネル# 1 (ハードディスク) やチャンネル# 3(ADPCM)のレジスタを読むと，妙な値が入っていますので注意してください。

チャンネル# 0 と# 1 は DMAC からの割り込みを禁止しており，ベクタには$0 F が入っています。FD や HD はコントローラ LSI のほうが割り込みを発生するため，DMAC に割り込みを発生させずに使っているわけです。

ベクタ$0 F の割り込みは非初期化割り込みベクタ番号と呼ばれ，68000 システムにおいて初期化の完了していない I/O デバイスから発生した割り込み番号として予約されているものです (HD 63450 はリセット後，NIV と EIV とともに$0 F に設定します)。

●図……21　Human 68 K での設定値

チャンネル／レジスタ	#0	#1	#2	#3
DCR	$80	$80	$08	$80
OCR	$B2	$B2	$00	$32
SCR	$04	$04	$00	$04
CCR	$00	$00	$00	$08
NIV	$0F	$0F	$68	$6A
EIV	$0F	$0F	$09	$6B
CPR	$00	$02	$03	$01

＊ベクタ番号 0

7 サンプルプログラム

DMACを操作するサンプルプログラムとして，テキスト画面のクリアを行うものと，グラフィック画面の矩形領域への転送を行うものを作成してみました。

サンプルプログラムはGCCやXCでコンパイル可能です。XCはシャープ純正ですが，実際には生成されるコードの質がよいことなどからフリーソフトウェアのGCCを利用されている方が多いと思われますので，サンプルはGCC用となっています。XCではvolatileが使用できないので，タイトルの最後にある# defineマクロをコメント内から出すか，リスト中のvolatileという文字列を削除してからコンパイルしてください。GCCを使用する場合には，逆にvolatileをつけておかないと，よけいなオプティマイズをされてしまい，動かなくなりますので削除しないようにしてください。

サンプルプログラム作成時に使用したバッチファイルは次のようなものです。

・GCC用

```
gcc -O -fomit-frame-pointer -finline-functions -fstrength-reduce %1 %2 %3 %4 %5 baslib.a iocslib.a doslib.a
```

・XC用

```
cc %1 %2 %3 %4 %5 /W /Y
```

7・1 DMACによるテキスト画面クリア

DMACを使用してテキスト画面クリアを行うプログラムをリスト1に示します。SUPER(0);でスーパバイザモードに入った後，テキストVRAMの先頭番地に0を書き込んでおきます。

DMA転送はMARの指す番地からDARの指す番地への転送で行っています。MARとDARをともにテキストVRAMの先頭番地にあわせ，DARだけをインクリメントするようにプログラムしておきます。これによって，VRAMの先頭番地のデータがテキストVRAM全体に書き込まれるわけです。このサンプルでは先頭番地に0を入れていますので，テキスト画面クリアになるわけです。

オペランドサイズはロングワード（32ビット）にしています。テキスト画面が256K

バイトあるのに対し，MTCは16ビット(64Kバイト)分しかないため，オペランドサイズをロングワードにして64K×4=256Kバイトを一度に転送するようにしたわけです。

●リスト……1　DMACによるテキスト画面クリア

```
/*
 * リスト1：DMAコントローラによるテキスト画面クリア
 *
 * XC ではvolatile がサポートされていないため、
 * 次の1行を入れてvolatileを無効にしてください
 *
 * #define  volatile
 */

#include <doslib.h>

struct  DMAREG {
        unsigned char   csr;
        unsigned char   cer;
        unsigned short  spare1;
        unsigned char   dcr;
        unsigned char   ocr;
        unsigned char   scr;
        unsigned char   ccr;
        unsigned short  spare2;
        unsigned short  mtc;
        unsigned char   *mar;
        unsigned long   spare3;
        unsigned char   *dar;
        unsigned short  spare4;
        unsigned short  btc;
        unsigned char   *bar;
        unsigned long   spare5;
        unsigned char   spare6;
        unsigned char   niv;
        unsigned char   spare7;
        unsigned char   eiv;
        unsigned char   spare8;
        unsigned char   mfc;
        unsigned short  spare9;
        unsigned char   spare10;
```

```
    unsigned char   cpr;
    unsigned short  spare11;
    unsigned char   spare12;
    unsigned char   dfc;
    unsigned long   spare13;
    unsigned short  spare14;
    unsigned char   spare15;
    unsigned char   bfc;
    unsigned long   spare16;
    unsigned char   spare17;
    unsigned char   gcr;
} ;

volatile struct DMAREG  *dma;
void main();
void dma_setup();
void dma_start();
void wait_complete();
void clear_flag();

void main()
{
    SUPER(0);
    *(unsigned int *)0xe00000 = 0;
    dma = (struct DMAREG *)0xe84080;  /* チャンネル#2を使用する */
    clear_flag();                     /* CSRのフラグ類をクリア  */
    dma_setup();                      /* DMAコントローラ初期化  */
    dma_start();                      /* 転送開始              */
    wait_complete();                  /* 転送終了待ち          */
    clear_flag();                     /* フラグ類をクリアしておく  */
}

void dma_setup()
{
    dma->dcr = 0x08;
    dma->ocr = 0x21;
    dma->scr = 0x01;
    dma->ccr = 0x00;
    dma->cpr = 0x03;
    dma->mfc = 0x05;
    dma->dfc = 0x05;
```

```
    dma->mtc = 0xffff;
    dma->mar = (unsigned char *)0xe00000;
    dma->dar = (unsigned char *)0xe00000;
}

void dma_start()
{
    dma->ccr |= 0x80;
}

void wait_complete()
{
    while(!(dma->csr & 0x90))
        ;
}

void clear_flag()
{
    dma->csr = 0xff;
}
```

❼·2 グラフィックVRAMへの矩形領域転送(その1)

不連続領域への転送が一度に行えるアレイチェインモードを利用して，グラフィック画面の矩形領域への転送を行うプログラムを作成してみました(リスト２)。65536色モードで画面にグラデーションパターンを書き込んだ後，先頭番地から順にバッファにデータを取り込みます。このバッファ上のデータを矩形領域に転送するような転送情報テーブルを配列上につくっています。転送情報１つで水平１ライン分の転送を行い，これを垂直方向のドット数分だけ並べて転送情報テーブルとしています。転送先のアドレスを順次変化させて，画面上では四角い領域が動いているように見せてみました。

●リスト……2　グラフィックVRAMへの矩形領域転送（アレイチェインモード）

```
/*
 * リスト2：アレイチェインモードによるグラフィック画面の矩形領域転送
 *
 * XC ではvolatile がサポートされていないため、
 * 次の1行を入れてvolatileを無効にしてください
 *
 * #define  volatile
 */

#include <doslib.h>

struct  DMAREG {
    unsigned char   csr;
    unsigned char   cer;
    unsigned short  spare1;
    unsigned char   dcr;
    unsigned char   ocr;
    unsigned char   scr;
    unsigned char   ccr;
    unsigned short  spare2;
    unsigned short  mtc;
    unsigned char   *mar;
    unsigned long   spare3;
    unsigned char   *dar;
    unsigned short  spare4;
    unsigned short  btc;
    unsigned char   *bar;
    unsigned long   spare5;
    unsigned char   spare6;
    unsigned char   niv;
    unsigned char   spare7;
    unsigned char   eiv;
    unsigned char   spare8;
    unsigned char   mfc;
    unsigned short  spare9;
    unsigned char   spare10;
    unsigned char   cpr;
    unsigned short  spare11;
    unsigned char   spare12;
    unsigned char   dfc;
```

```
    unsigned long   spare13;
    unsigned short  spare14;
    unsigned char   spare15;
    unsigned char   bfc;
    unsigned long   spare16;
    unsigned char   spare17;
    unsigned char   gcr;
} ;

struct XFR_INF {
    unsigned short *adrs;
    unsigned short length;
} xfr_inf[512];
unsigned short  databuf[256*256];
volatile struct DMAREG  *dma;
unsigned short  src_data;

void main();
void init_screen();
void dma_box();
void dma_setup();
void dma_start();
void wait_complete();
void clear_flag();

void main()
{
    int i;
    screen (1,3,1,1);
    SUPER(0);
    init_screen();
    for (i = 0; i<255; i+=4)
        dma_box(databuf,255-i,i,511-i,i+256,0xffff);
}

void init_screen()
{
    unsigned short  *vram,*buf;
    unsigned int    i,h,s,v;
    vram = (unsigned short *)0xc00000;
    for (i=0; i<512*512; i++) {
```

```
        s = i & 0x1f;
        v = (i >> 5) & 0x1f;
        h = ((i >> 10) % 0xc0);
        *vram++ = hsv(h, s, v);
    }
    vram = (unsigned short *)0xc00000;
    buf = databuf;
    for (i=0; i<256*256; i++)
        *buf++ = *vram++;
}

void dma_box(buf, x1, y1, x2, y2, col)
    unsigned short  *buf;
    unsigned int    x1, y1, x2, y2, col;
{
    int i, xlen, ylen;
    unsigned short *sadrs;
    xlen = x2-x1;
    ylen = y2-y1;
    src_data = col;
    sadrs = (unsigned short *)0xc00000;
    sadrs += 512*y1+x1;
    for(i=0; i <= ylen; i++, sadrs+=512) {
        xfr_inf[i].adrs = sadrs;
        xfr_inf[i].length = xlen;
    }
    dma = (struct DMAREG *)0xe84080;
    clear_flag();
    dma_setup(buf, ylen+1);
    dma_start();
    wait_complete();
    clear_flag();
}

void dma_setup(bufadrs, links)
    unsigned short  *bufadrs;
    unsigned int    links;
{
    dma->dcr = 0x08;
    dma->ocr = 0x99;
    dma->scr = 0x05;
```

```
    dma->ccr = 0x00;
    dma->cpr = 0x03;
    dma->mfc = 0x05;
    dma->dfc = 0x05;
    dma->bfc = 0x05;

    dma->btc = links;
    dma->dar = (unsigned char *)bufadrs;
    dma->bar = (unsigned char *)xfr_inf;
}

void dma_start()
{
    dma->ccr |= 0x80;
}

void wait_complete()
{
    while(!(dma->csr & 0x90))
        ;
}

void clear_flag()
{
    dma->csr = 0xff;
}
```

❼·3 グラフィックVRAMへの矩形領域転送(その2)

7-2で行った矩形領域への転送を，リンクアレイチェインモードを使用するように書き換えたのがリスト3です。リスト2と比較すると，アレイチェインモードとリンクアレイチェインモードの違いがわかると思います。

●リスト……3　グラフィックVRAMへの矩形領域転送（リンクアレイチェインモード）

```
/*
 * リスト3：リンクアレイチェインモードによるグラフィック画面の矩形領域転送
 *
 * XC ではvolatile がサポートされていないため、
 * 次の1行を入れてvolatileを無効にしてください
 *
 * #define  volatile
 */

#include <doslib.h>

struct  DMAREG {
    unsigned char   csr;
    unsigned char   cer;
    unsigned short  spare1;
    unsigned char   dcr;
    unsigned char   ocr;
    unsigned char   scr;
    unsigned char   ccr;
    unsigned short  spare2;
    unsigned short  mtc;
    unsigned char   *mar;
    unsigned long   spare3;
    unsigned char   *dar;
    unsigned short  spare4;
    unsigned short  btc;
    unsigned char   *bar;
    unsigned long   spare5;
    unsigned char   spare6;
    unsigned char   niv;
    unsigned char   spare7;
    unsigned char   eiv;
    unsigned char   spare8;
    unsigned char   mfc;
    unsigned short  spare9;
    unsigned char   spare10;
    unsigned char   cpr;
    unsigned short  spare11;
    unsigned char   spare12;
    unsigned char   dfc;
    unsigned long   spare13;
    unsigned short  spare14;
```

```
    unsigned char   spare15;
    unsigned char   bfc;
    unsigned long   spare16;
    unsigned char   spare17;
    unsigned char   gcr;
};

struct XFR_INF {
    unsigned short *adrs;
    unsigned short length;
    struct  XFR_INF *link;
} xfr_inf[512];
unsigned short  databuf[256*256];
volatile struct DMAREG  *dma;
unsigned short  src_data;

void main();
void init_screen();
void dma_box();
void dma_setup();
void dma_start();
void wait_complete();
void clear_flag();

void main()
{
    int i;
    screen (1,3,1,1);
    SUPER(0);
    init_screen();
    for (i = 0; i<255; i+=4)
        dma_box(databuf,255-i,i,511-i,i+256,0xffff);
}

void init_screen()
{
    unsigned short  *vram,*buf;
    unsigned int    i,h,s,v;
    vram = (unsigned short *)0xc00000;
    for (i=0; i<512*512; i++) {
        s = i & 0x1f;
        v = (i >> 5) & 0x1f;
        h = ((i >> 10) % 0xc0);
```

```
            *vram++ = hsv(h, s, v);
        }
        vram = (unsigned short *)0xc00000;
        buf = databuf;
        for (i=0; i<256*256; i++)
            *buf++ = *vram++;
}

void dma_box(buf, x1, y1, x2, y2, col)
    unsigned short  *buf;
    unsigned int     x1, y1, x2, y2, col;
{
    int i, xlen, ylen;
    unsigned short *sadrs;
    xlen = x2-x1;
    ylen = y2-y1;
    src_data = col;
    sadrs = (unsigned short *)0xc00000;
    sadrs += 512*y1+x1;
    for(i=0; i <= ylen; i++, sadrs+=512) {
        xfr_inf[i].adrs = sadrs;
        xfr_inf[i].length = xlen;
        xfr_inf[i].link = &xfr_inf[i+1];
    }
    xfr_inf[i-1].link = 0;
    dma = (struct DMAREG *)0xe84080;
    clear_flag();
    dma_setup(buf);
    dma_start();
    wait_complete();
    clear_flag();
}

void dma_setup(bufadrs)
    unsigned short  *bufadrs;
{
    dma->dcr = 0x08;
    dma->ocr = 0x9d;
    dma->scr = 0x05;
    dma->ccr = 0x00;
    dma->cpr = 0x03;
```

```
    dma->mfc = 0x05;
    dma->dfc = 0x05;
    dma->bfc = 0x05;
    dma->dar = (unsigned char *)bufadrs;
    dma->bar = (unsigned char *)xfr_inf;
}

void dma_start()
{
    dma->ccr |= 0x80;
}

void wait_complete()
{
    while(!(dma->csr & 0x90))
        ;
}

void clear_flag()
{
    dma->csr = 0xff;
}
```

割り込み

X 68000 ではシステムの状態変化や LSI からのサービス要求のほとんどは割り込みによって通知されます。ここでは，割り込み動作の概要や Human 68 K における割り込みベクタの一覧などについて説明します。

1 割り込み系統とレベル割り付け

X 68000 の割り込み系統図を 72 ページ図 1 に示します。

X 68000 の CPU である 68000 は，割り込みにレベル 1 からレベル 7 までの 7 つの優先順位を与えており，外部回路は，要求する割り込みレベルを CPU の IPL 0，IPL 1，IPL 2 の 3 本の信号線を使って知らせます。レベル 0 は割り込みがない状態を示すのに使用されるため，優先度は 7 レベルまでとなるわけです。7 つのレベルの割り込みのうち，もっとも優先順位の低いのがレベル 1 で，もっとも高い割り込みがレベル 7 となっています。CPU のステータスレジスタには 3 ビットの割り込みマスクビットがあり，この値以下の割り込みはマスクされます。CPU が割り込みを受け付けると，その割り込みレベルが自動的にマスクビットに反映され，優先順位がより高い割り込みだけが入り込めるようになるわけです。ただし，レベル 7 の割り込みだけは例外で，ステータスレジスタのマスクビットによってマスクされません。このことから，レベル 7 の割り込みは NMI (Non Maskable Interrupt) とも呼ばれます。

X 68000 では，この 7 つのレベルを次のように割り振っています。

●図……1　割り込み系統図

NMIスイッチ
MFP(MC68901)
GPIP7
GPIP6
GPIP5
GPIP4
GPIP3
GPIP2
GPIP1
GPIP0
CRTC H-SYNC
CRTC IRQ
(無接続)
CRTC V-DISP
FM音源
POWERスイッチ
EXPON
RTC ALARM
CPU
IPL2
IPL1
IPL0
プライオリティ
エンコーダ
プライオリティ
(Level7)
高
低
プライオリティ
(Level 1)
タイマ
A～D
シリアル
入出力
キーボード
SCC
チャンネルA
(Z8530)
チャンネルB
RS-232C
マウス
DMAC
(HD63450)
ADPCM
(拡張スロット)
HD
FD
FDC
(μPD72065)
FDD
I/Oコントローラ
(シャープ独自
開発ASIC)
HD
コントローラ
ハードディスク
プリンタ
インタフェース
プリンタ
(拡張スロット)

- レベル7（NMI）：本体上のNMIスイッチ
- レベル6　：MFP（マルチファンクションペリフェラル）
  ［CRTC，FM音源，タイマ，キーボードなど］
- レベル5　：SCC（シリアルコミュニケーションコントローラ）
  ［RS-232C，マウス］
- レベル4　：拡張スロット
- レベル3　：DMAC（DMAコントローラ）
  ［ADPCM，FD，HD］
- レベル2　：拡張スロット
- レベル1　：I/OコントローラLSI
  ［FD，HD，プリンタ］

●2 割り込み動作

68000の割り込み応答動作の概略を74ページの図2に示します。

周辺デバイスが割り込み要求を発生すると，①外部回路で優先順位のデコードを行い，IPL 0〜2の3本の信号線で，そのレベルをCPUに通知します②。CPUは割り込みを受け付けると，アドレスバスの下位3ビット（A 1〜A 3）に受け付けた割り込みレベルを出力し，同時にファンクションコード（FC 0〜FC 2）をすべて'H'レベルにして割り込みへの応答サイクルであることを示し，周辺デバイスから割り込みベクタを読み出しにいきます③。

周辺デバイスは，データバスの下位8ビットに割り込みベクタを出力し④，DTACK信号でCPUに対して有効な割り込みベクタがデータバス上に乗っていることを示します⑤。CPUは，このベクタを読み取り，割り込み処理ルーチンへの移行を始めるわけです。

周辺デバイスがオートベクタを指定する(DTACK信号のかわりにVPA信号をアクティブにする)と⑥，CPUはベクタの読み出しを行わず，各レベルに応じたデフォルトのベクタである$19〜$1 F（それぞれレベル1〜レベル7に対応する）を使用します。

Human 68 Kは，レベル7のNMIがオートベクタの$1 Fを使用するほかは，すべて周辺デバイスがベクタを出力するようにして使っています。

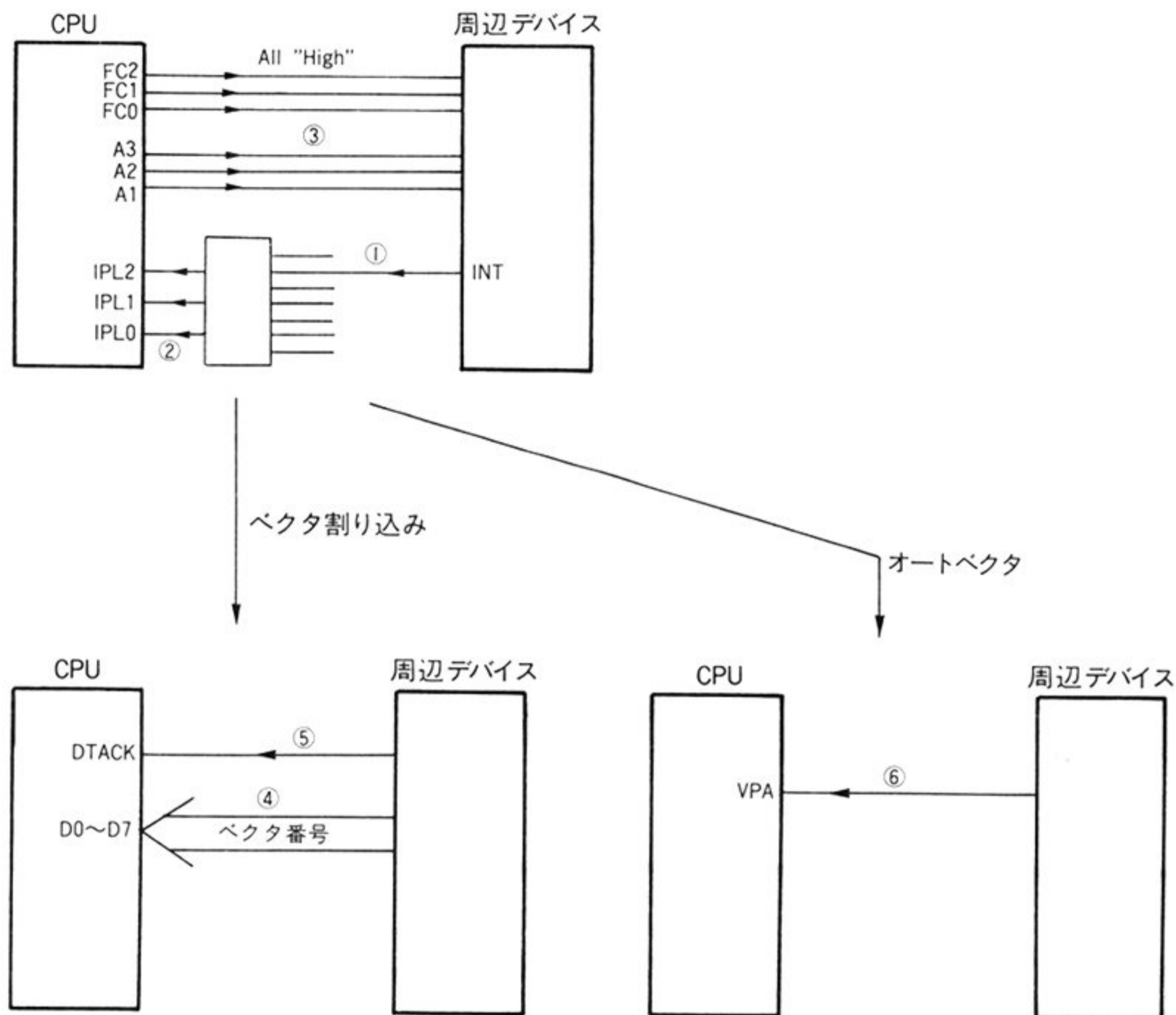
●図……2　68000の割り込み動作

●3 例外ベクタ

図3に68000の例外ベクタとHuman 68Kにおける設定，利用のされ方を示します。これらのベクタのうち，$00〜$3Fまでは CPU デザインを行ったメーカ（モトローラ）によって予約されている領域であり，周辺デバイスで割り込みベクタとして使用することは禁止されています。Human 68Kは，$40〜$4FをMFP，$50〜$5FをSCC，$60〜$63をI/Oコントローラ，$64〜$6BをDMACに割り付けています。

●図……3　例外ベクタの割り当て

ベクタ番号		ベクタテーブルアドレス	ベクタの割り当て	Human68Kでの使用
10進	16進			
0	$00	$000000	リセット後のSSPの値	
1	$01	$000004	〃　PC　〃	
2	$02	$000008	バスエラー	
3	$03	$00000C	アドレスエラー	
4	$04	$000010	不当命令	
5	$05	$000014	ゼロによる除算	
6	$06	$000018	CHK命令	
7	$07	$00001C	TRAPV命令	
8	$08	$000020	特権違反	
9	$09	$000024	トレース	
10	$0A	$000028	ライン1010エミュレータ	SX-Window用SXコール
11	$0B	$00002C	ライン1111エミュレータ	浮動小数点演算
12	$0C	$000030	未使用(将来拡張用)	
13	$0D	$000034	〃	
14	$0E	$000038	〃	
15	$0F	$00003C	未初期化割り込み	
16～23	$10～$17	$000040～05C	未使用(将来拡張用)	
24	$18	$000060	スプリアス割り込み	
25	$19	$000064	レベル1割り込み(オートベクタ時)	
26	$1A	$000068	〃　2　〃	
27	$1B	$00006C	〃　3　〃	
28	$1C	$000070	〃　4　〃	
29	$1D	$000074	〃　5　〃	
30	$1E	$000078	〃　6　〃	
31	$1F	$00007C	〃　7　〃	NMIスイッチ
32～39	$20～$27	$000080～09C	TRAP0～TRAP7命令	
40	$28	$0000A0	TRAP8命令	システム予約
41	$29	$0000A4	〃　9　〃	DB.Xのブレークポイント
42	$2A	$0000A8	〃　A　〃	POWER OFF/リセット処理
43	$2B	$0000AC	〃　B　〃	BREAKキーによるHDOFF等
44	$2C	$0000B0	〃　C　〃	COPYキーによるハードコピー等
45	$2D	$0000B4	〃　D　〃	CTRL+Cによるブレークチェックフラグセット
46	$2E	$0000B8	〃　E　〃	エラー表示(中止/再実行/無視の選択)
47	$2F	$0000BC	〃　F　〃	IOCSコール
48～63	$30～$3F	$0000C0～0FC	未使用(将来拡張用)	
64～79	$40～$4F	$000100～13C	ユーザ用割り込みベクタ	MFP
80～95	$50～$5F	$000140～17C	〃	SCC
96～99	$60～$63	$000180～18C	〃	I/Oコントローラ
100～107	$64～$6B	$000190～1AC	〃	DMAC
108～255	$6C～$FF	$0001B0～3FC	〃	未使用

4 割り込みベクタ設定ポート

周辺デバイスごとに割り込みベクタを設定するポートを探すのは面倒ですので，図4に各周辺デバイスごとに割り込みベクタを設定するポートをまとめておきました。

●図……4　割り込みベクタの設定ポート

LSI			アドレス	bit7 … bit0
MFP			$E88017	P (上位4ビット) ／ 割り込み要因で変化*1 (下位4ビット)
SCC			$E98003/7 (書き込みレジスタ2)	P
DMAC	CH #0	NIV	$E84025	P
		EIV	$E84027	P
	CH #1	NIV	$E84065	P
		EIV	$E84067	P
	CH #2	NIV	$E840A5	P
		EIV	$E840A7	P
	CH #3	NIV	$E840E5	P
		EIV	$E840E7	P
I/Oコントローラ			$E9C003	P (上位6ビット) ／ 割り込み要因で変化*2 (下位2ビット)

P：任意設定可

＊1:0000:GPIP0
0001:GPIP1
0010:GPIP2
0011:GPIP3
0100:タイマD
0101:タイマC
0110:GPIP4
0111:GPIP5
1000:タイマB
1001:送信エラー
1010:送信バッファ空
1011:受信エラー
1100:受信バッファフル
1101:タイマA
1110:GPIP6
1111:GPIP7

＊2:00:FDC
01:FDD
10:HD
11:プリンタ

MFP

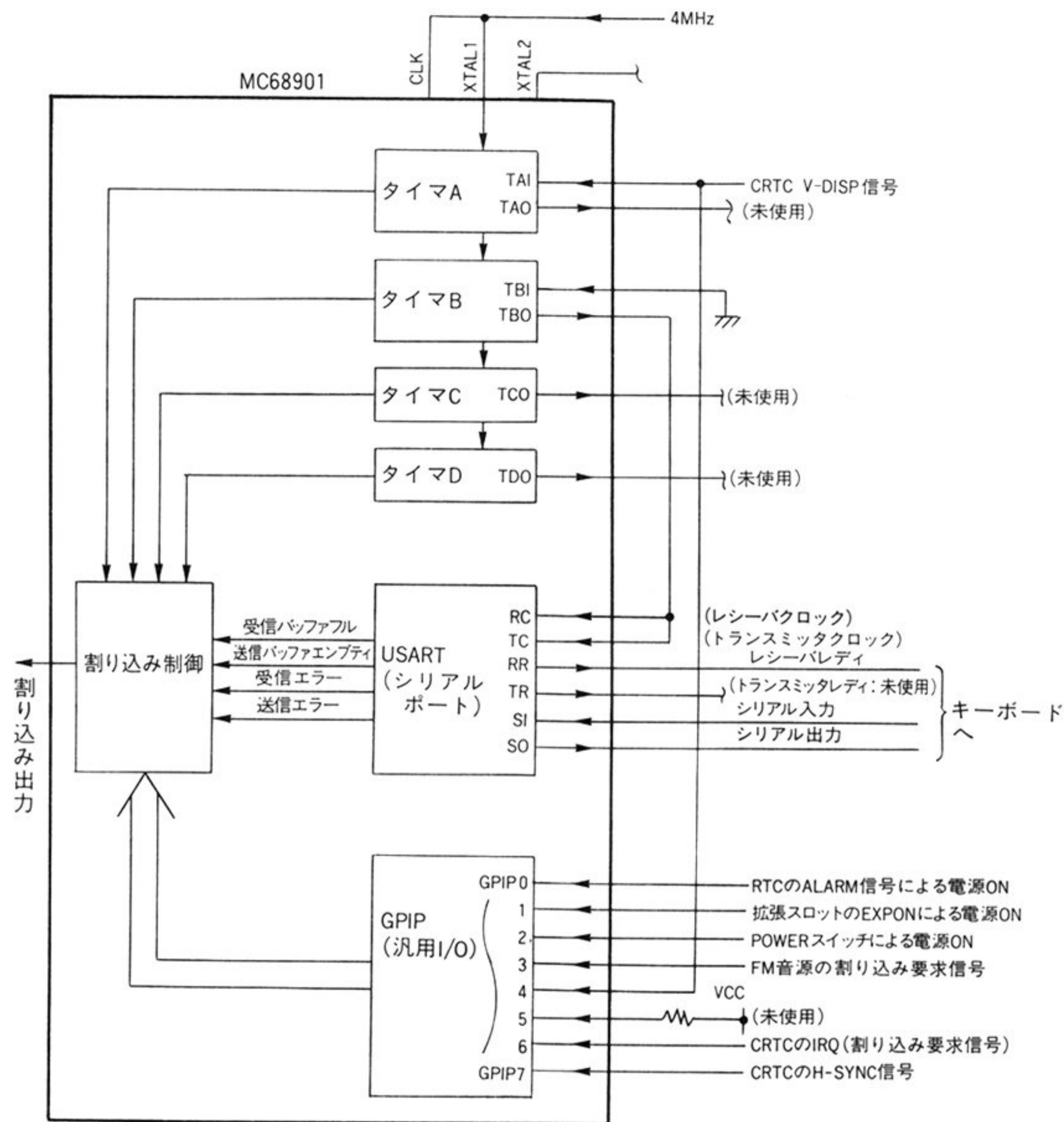
タイマや汎用I/O，シリアルポートなどを1チップにまとめあげたMFPは，キーボードのほか，定周期に発生するタイマ割り込み，CRTCやFM音源，RTCのアラーム信号など，雑多なステータスの取り込みに使用されています。

1 概要

MFP(マルチファンクションペリフェラル　MC 68901)は，カウンタ/タイマ，シリアルポートや汎用I/Oポートなどを1つのLSIの中に入れたものです。78ページの図1にMFPの内部ブロック図とX68000での接続状態の概略を示します。MFP内部は，4つのタイマ，1チャンネルのシリアルポート，8ビット分の汎用I/Oポートを持っており，X 68000はCRTCからの割り込みや電源ONの要因判別，キーボードとのインタフェースなどに使用しています。

2 MFPの各機能の割り付け

MFPの持つ各機能をX 68000ではどのように割り付けているか，かんたんに見ておくこと

●図……1　MFPの内部ブロック図

にしましょう。

4つのタイマのうち，タイマBはキーボードとの通信を行うシリアルポートの伝送速度を決めるクロックとして使用されていますので，設定や動作モードを変更したりすると，キーボードが使えなくなってしまいます。その他のタイマはハード的には用途は指定されていません。Human 68 KではタイマCをカーソルの点滅やFDDのモータ停止タイミングの作成などに，タイマDはVersion 2.0以降で疑似マルチタスク動作用として使用しています。

タイマAの制御線であるTAI入力には，CRTCが出力するV-DISP（垂直表示期間）信号が入っていますので，V-DISP信号の変化した回数をカウントして，一定回数ごとにCPUに

割り込みをかけるようにしたり，V-DISP信号の周期の測定を行うことも可能です。

MFPのシリアルポートはいくつもの動作モードを持っていますが，X 68000では接続する相手がキーボードに限定されていますので，キーボードの通信モードにあわせた設定で使うことになります。

GPIP 0～GPIP 7の8つの汎用I/Oポートのうち，未使用となっているGPIP 5以外はすべて入力ポートとして使われています。GPIP 5は外部でHレベルに固定されていますので，リードすると，つねに'1'が読み出されます。

3 MFPのレジスター覧

MFPのレジスタの一覧を80ページの図2に示します。

MFPのレジスタは$E 88001～$E 8802 F番地に配置されています。レジスタはすべて8ビット長であるため，奇数番地(ワードアクセス時の下位バイト)のみとなります。MFPのレジスタのうち，GPIPの制御に使われるのが$E 88001～$E 88005，割り込み制御に使われるのが$E 88007～$E 88017，タイマ制御用が$E 88019～$E 88025，USART (シリアルポート) 制御用が$E 88027～$E 8802 Fとなっています。

MFPのレジスタは，ステータス入力や一部の特殊な機能を持たせたもの以外は基本的にすべてライト/リードとも可能となっています。図の中で斜線が引いてあるビットは未使用です。未使用ビットはリードすると'0'が読み出されます。ライト時は'1'，'0'のいずれでもかまいませんが，とくに意味のないかぎり，他のLSIなどと同様，'0'にしておくのが普通でしょう。

4 GPIP(汎用I/Oポート)

GPIPの制御に関係するレジスタのビット配置を81ページの図3に示します。GPIPの制御用のレジスタは，GPIP，AER，DDRの3つがありますが，どれも同じビット配置ですので，図は1つにまとめておきました。

●図……2　MFPのレジスター覧

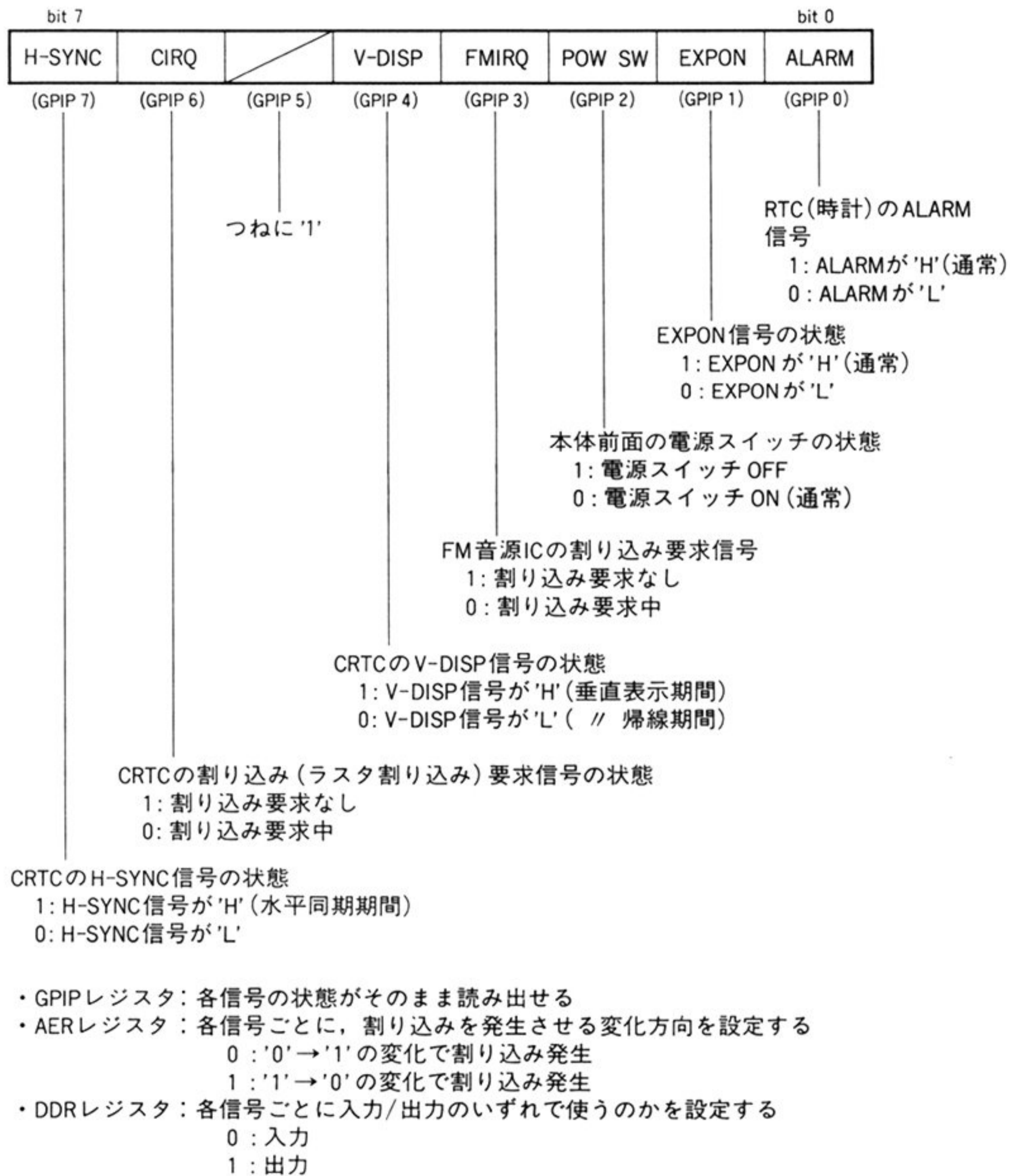
種　別	アドレス	略称	bit 7							bit 0	レジスタ名
GPIP制御	$E88001	GPIP	GPIP 7	GPIP 6	GPIP 5	GPIP 4	GPIP 3	GPIP 2	GPIP 1	GPIP 0	汎用I/Oレジスタ
	3	AER	GPIPと同様								アクティブエッジレジスタ
	5	DDR	GPIPと同様								データ方向レジスタ
割り込み制御	$E88007	IERA	GPIP 7	GPIP 6	タイマA	バッファフル	受信エラー	バッファエンプティ	送信エラー	タイマB	割り込みイネーブルレジスタA
	9	IERB	GPIP 5	GPIP 4	タイマC	タイマD	GPIP 3	GPIP 2	GPIP 1	GPIP 0	割り込みイネーブルレジスタB
	B	IPRA	IERAと同様								割り込みペンディングレジスタA
	D	IPRB	IERBと同様								割り込みペンディングレジスタB
	F	ISRA	IERAと同様								割り込みインサービスレジスタA
	$E88011	ISRB	IERBと同様								割り込みインサービスレジスタB
	3	IMRA	IERAと同様								割り込みマスクレジスタA
	5	IMRB	IERBと同様								割り込みマスクレジスタB
	7	VR	V7	V6	V5	V4	S				ベクタレジスタ
タイマ制御	$E88019	TACR				リセットTAO	AC3	AC2	AC1	AC0	タイマAコントロールレジスタ
	B	TBCR				リセットTAO	BC3	BC2	BC1	BC0	タイマBコントロールレジスタ
	D	TCDCR		CC2	CC1	CC0		DC2	DC1	DC0	タイマCコントロールレジスタ
	F	TADR	D7	D6	D5	D4	D3	D2	D1	D0	タイマAデータレジスタ
	$E88021	TBDR	TADRと同様								タイマBデータレジスタ
	3	TCDR	TADRと同様								タイマCデータレジスタ
	5	TDDR	TADRと同様								タイマDデータレジスタ
USART制御	$E88027	SCR	D7	D6	D5	D4	D3	D2	D1	D0	SYNCキャラクタレジスタ
	9	UCR	CLK	WLI	WLO	STI	STO	PE	E/O		USARTコントロールレジスタ
	B	RSR	BF	OE	PE	FE	F/S or B	M/CIP	SS	RE	レシーバステータスレジスタ
	D	TSR	BE	UE	AT	END	B	H	L	TE	トランスミッタステータスレジスタ
	F	UDR	D7	D6	D5	D4	D3	D2	D1	D0	USARTデータレジスタ

❹·1　GPIPレジスタ

GPIPレジスタは，GPIP 0～GPIP 7の各ビットの状態を読み出したり，出力データを書き込むレジスタです。X 68000ではGPIP 0～GPIP 7のすべてを入力として使いますので，このレジスタはリードのみとなります。次に各GPIPビットに接続されている信号の説明をしておきましょう。

GPIP 7にはCRTCのH-SYNC (水平同期) 信号が接続されています。'1'でCRTCが水平同期期間であることを示します。

●図……3　GPIP, AER, DDRのビット配置($E 88001, $E88003, $E88005)

・GPIPレジスタ：各信号の状態がそのまま読み出せる
・AERレジスタ：各信号ごとに，割り込みを発生させる変化方向を設定する
　0：'0'→'1'の変化で割り込み発生
　1：'1'→'0'の変化で割り込み発生
・DDRレジスタ：各信号ごとに入力/出力のいずれで使うのかを設定する
　0：入力
　1：出力

GPIP 6はCRTCのラスタ割り込み信号が接続されています。'0'でCRTCがラスタ割り込み要求を発生していることを示します。ラスタ割り込み機能の詳細は，CRTCの説明の章を参照してください。

GPIP 5は未使用ビットです。外部でHレベルに固定されているため，GPIP 5は，つねに'1'が読み出されます。

GPIP 4はCRTCのV-DISP(垂直表示期間)信号が接続されています。'1'で垂直表示期間であることを，'0'で垂直帰線期間であることを示します。

GPIP 3はFM音源ICからの割り込み要求信号です。'0'でFM音源からの割り込み要求が発生していることを示します。

GPIP 2, 1, 0はX 68000の電源がONになる要因が接続されています。X 68000は，本体正面の電源スイッチによる通常のON/OFFのほか，拡張スロットのリモート電源ON信号や本体背面のリモート端子(この両者は同じ信号として扱われています)，RTC(リアルタイムクロック：時計)のALARM信号などで電源を入れることができるようになっています。

このため，X 68000では電源がONとなった要因をソフトウェアで判定できるようにしているのです。本体正面の電源スイッチがONになっているとGPIP 2が，拡張スロットやリモート端子が'L'(電源ON状態)になっているとGPIP 1，リアルタイムクロックのALARM信号がALARM状態になっているとGPIP 0が，それぞれ'0'になります。

❹·2 AER(アクティブエッジレジスタ)

GPIPは，どのビットも'0'から'1'，あるいは'1'から'0'への変化で割り込みを発生することができるようになっています。AERは，各ビットごとにいずれの変化で割り込みを発生するかを指定するレジスタです。'1'にすると'0'から'1'への変化で，'0'にすると'1'から'0'への変化で割り込みを発生するようになります。AERのGPIP 3とGPIP 4のビットは，タイマの制御信号の割り込みのエッジ設定と兼用になっています。この点については，タイマのところで説明します。

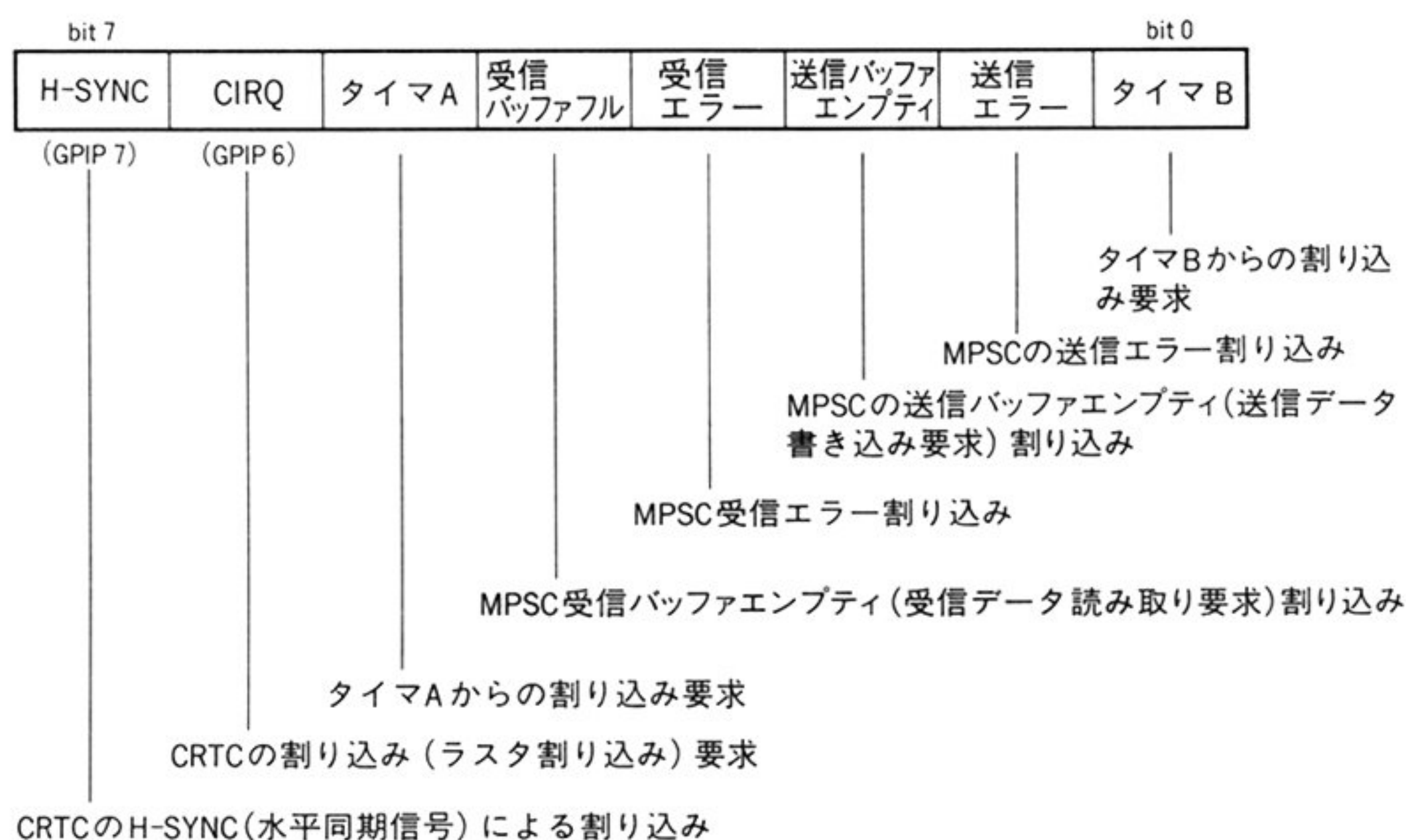
❹·3 DDR(データディレクションレジスタ)

GPIPの各ビットごとに入力として使うか，出力として使うかを設定するレジスタです。'1'で出力，'0'で入力となります。

X 68000では，GPIPはすべて入力ポートとして使いますので，DDRは全ビットとも'0'を設定します。

●5 割り込み制御

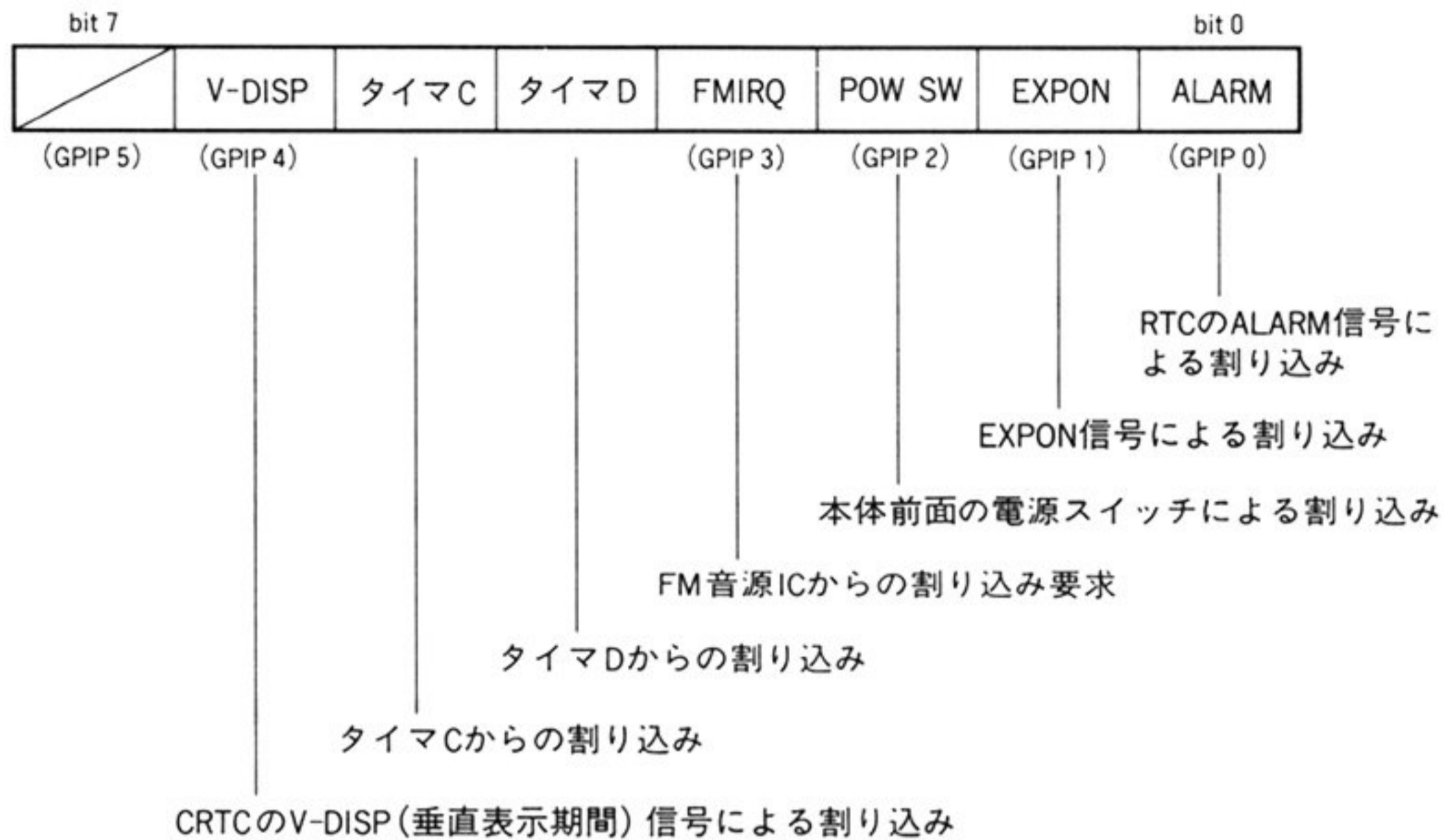
MFPの割り込み制御に関係するレジスタを図4，図5および図6に示します。
MFPは16種類の割り込み要因を持っており，これが8ビット×2本のレジスタに配分されています。割り込みの優先順位は固定で，GPIP 7 (H-SYNC) がもっとも高く，以下，レジスタのビット並びどおり GPIP 6 (CIRQ)，タイマA……と続き，GPIP 0 (ALARM) がもっとも低くなっています。

●図……4　IERA, IPRA, ISRA, IMRA($E 88007, $E 8800 B, $E 8800 F, $E 88013)

・IERA：割り込み発生の許可/禁止を制御する
　1：割り込み発生許可
　0：　〃　禁示

・IPRA：割り込み要求がペンディング(保留)されていることを示す
　1：割り込み要求がペンディングされている
　0：　〃　されていない

・ISRA：割り込み要求が処理中(インサービス)である
　1：割り込み要求は処理中である
　0：　〃　ではない

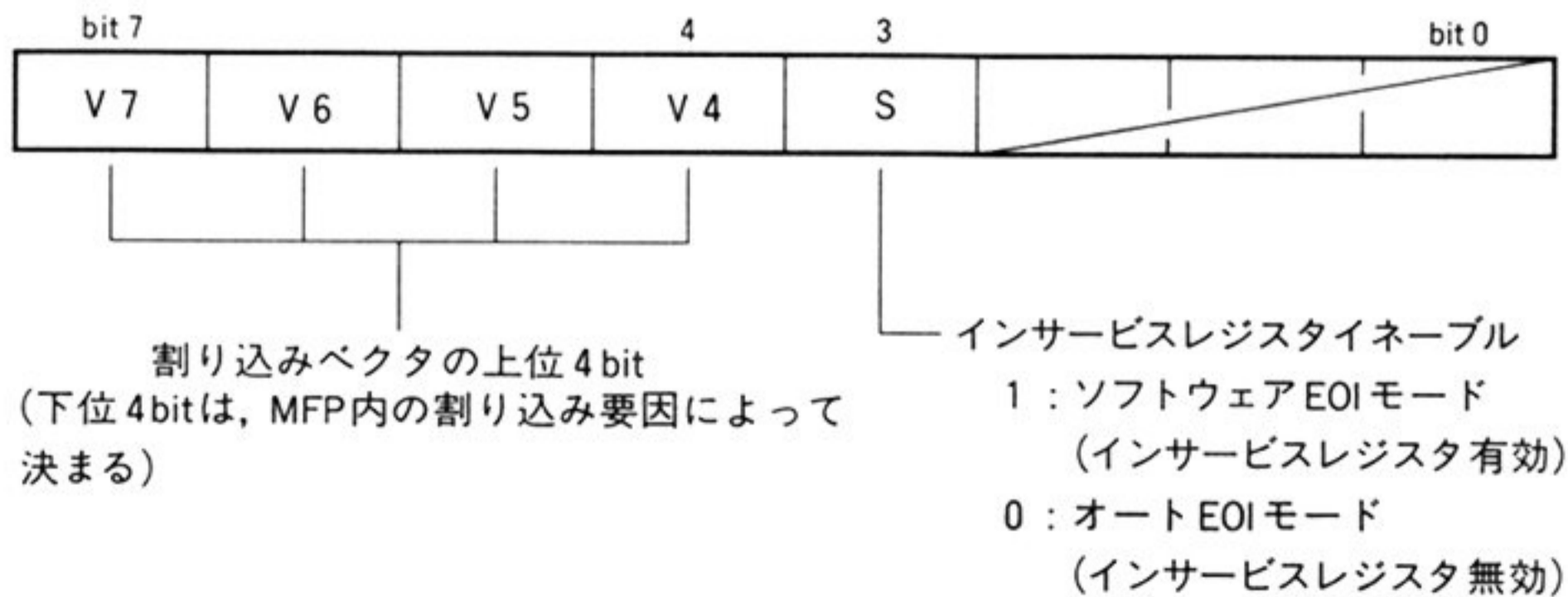
・IMRA：割り込みマスクの制御を行う
　1：割り込み要求をマスクしない(割り込み発生可)
　0：　〃　する　(　〃　不可)

●図……5　IERB, IPRB, ISRB, IMRB($E 88009, $E 8800 D, $E 88011, $E 88015)

・IERB：割り込み発生の許可/禁止を制御する
　1：割り込み発生許可
　0：　〃　禁止

・IPRB：割り込み要求がペンディング(保留)されていることを示す
　1：割り込み要求はペンディングされている
　0：　〃　されていない

・ISRB：割り込み要求が処理中(インサービス)であることを示す
　1：割り込み要求は処理中である
　0：　〃　ではない

・IMRB：割り込みマスクの制御を行う
　1：割り込み要求をマスクしない(割り込み発生可)
　0：　〃　する　(　〃　不可)

●図……6　VR(ベクタレジスタ)$E 88017

❺·1 IERA/IERB(割り込みイネーブルレジスタA/B)

IERA/IERBは，割り込み発生の許可/禁止を制御するレジスタです。'1'にすると該当する信号による割り込みの発生が許可され，'0'にすると禁止されます。

❺·2 IPRA/IPRB(割り込みペンディングレジスタA/B)

IPRA/IPRBは，割り込み要求がペンディング(保留)されていることを示すレジスタです。IPRA/IPRBは,MFPが割り込み要求のきたことを認識すると'1'となり,CPUに該当する割り込み要求が伝えられた(割り込みベクタを渡した)ときに'0'に復帰します。つまり，'1'が立っている状態は，割り込み要求が発生したものの，まだCPUに割り込みとして伝わっていないということを示しているわけです。

IPRA/IPRBの各ビットは，IERA/IERBによって割り込みの発生が禁止されたり，CPUがIPRA/IPRBの該当ビットに'0'を書き込むことによっても'0'になります。

❺·3 ISRA/ISRB(インサービスレジスタA/B)

該当する割り込みがサービス(処理)中であることを示すレジスタです。MFPからCPUに対して割り込みが伝えられる(CPUにベクタを引き渡す)と,該当するビットが'1'になり,CPUが該当するビットを'0'にしたデータをISRA/ISRBレジスタに書き込むと'0'になります。

MFPは,このようなソフトウェアによるサービス終了通知(EOI：End Of Interruptと呼びます)のほか，自動EOIモードにプログラムすることもできます。このとき，MFPはCPUにベクタを渡した時点でサービス終了とみなしますので，ISRA/ISRBの該当ビットも即座に'0'に復帰します。

ISRA/ISRBが'0'でIPRA/IPRBが'1'になると，MFPは該当するビットの割り込み要求を行います。つまり，自動EOIモードの場合には，連続して同一の割り込みが入ってくることも可能であるわけです。

MFPをソフトウェアEOIで動作させるか，自動EOIで動作させるかはベクタレジスタで設定します。詳細は，ベクタレジスタの説明を見てください。

❺·4 IMRA/IMRB（インタラプトマスクレジスタA/B）

割り込みのマスク制御を行うレジスタです。'1'だと割り込み発生が可能になります。IERA/IERBレジスタとよく似たレジスタです。両者の違いは，割り込みの発生を禁止（'0'を設定）している間に新たな割り込みが入ったときにMFPがどのように振る舞うかにあります。

IERA/IERBが'0'になっていると，この間の割り込み要求は完全に無視されます。IMRA/IMRBは，たとえ'0'になっていても，IERA/IERBが'1'になってさえいれば，MFPは割り込み要求を受け取り，IPRA/IPRBの該当ビットを'1'にします。その後，IMRA/IMRBの該当ビットが'1'になった時点でCPUに対して割り込みを発生します。

IERA/IERBは割り込み要求の発生元を抑えてしまうもの，IMRA/IMRBはMFPからの割り込み要求出力を抑えるだけのものと考えるとわかりやすいかもしれません。

❺·5 ベクタレジスタ

MFPがCPUに割り込み要求をかけるときに出力する，ベクタ番号の設定などを行うレジスタです。出力される8ビットのベクタのうち，上位4ビットをレジスタのビット4からビット7で設定します。ベクタの下位4ビットは，MFPの割り込み優先度と同じ順序になっており，'1111'がもっとも優先度の高いGPIP 7で，以下，GPIP 6，タイマA……と続き，もっとも優先度の低いGPIP 0が'0000'となっています。

ベクタレジスタのSビットは，割り込みに対するEOIのモードをソフトウェアEOIとするか，自動EOIにするかを選択するビットです。

このビットを'1'にするとソフトウェアEOIモードとなり，インサービスレジスタの該当ビットはCPUによる割り込み受付後，EOI処理（ISRA/ISRBの該当ビットに'0'を書き込む）が行われるまで'1'となり，割り込みがサービス中であることを示すのに使用されます。

Sビットに'0'を設定すると自動EOIモードとなり，割り込み要求がCPUに受け付けられた時点でEOIされたものとみなしますので，ISRA/ISRBの各ビットは意味を持たなくなります。

6 タイマ

MFPは，タイマAからタイマDまでの4つのタイマを持っています。このうち，タイマCとDは，単純に入力された周波数を1/Nに分周するディレイモード動作しかできませんが，タイマAとタイマBは，専用の入力端子(TAI/TBI)を利用して，入力端子の状態が変化する間隔の測定（パルス幅測定モード）や，変化の回数のカウント（イベントカウントモード）などを行わせることもできるようになっています。

6·1 タイマの動作モード

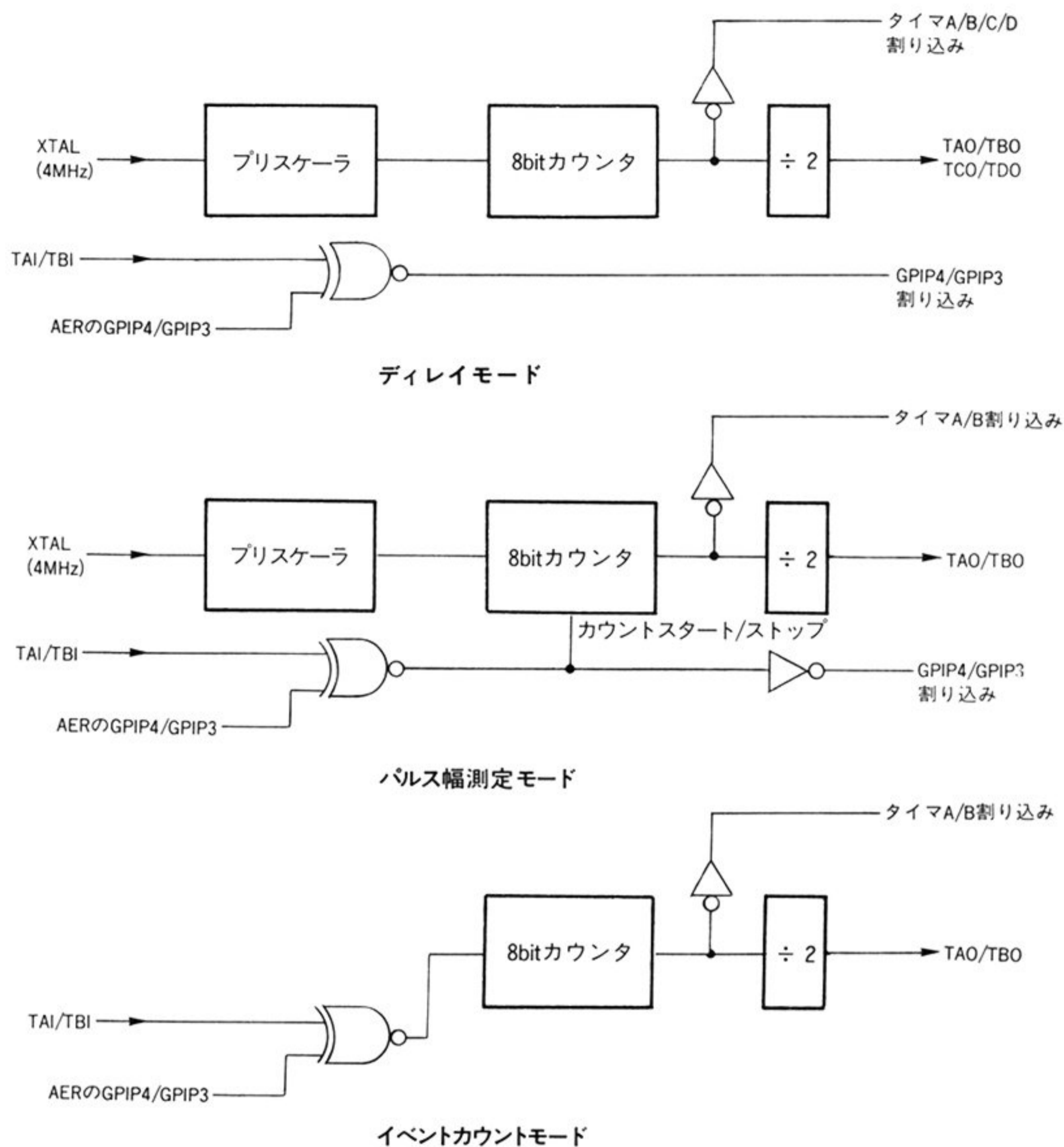
MFPのタイマが持つ動作モードの概略を88ページの図7に示します。図の中で8ビットカウンタとなっているところがCPUによって値を読み書きすることのできるカウンタで，このレジスタのアクセスによって任意の周波数を得たり，経過時間やイベントの回数の読み取りを行います。この各動作モードについて説明しておくことにしましょう。

6·1·1 ディレイモード

ディレイモードは，任意の周波数を得たり，一定周期で割り込みを発生するような用途に使用されるモードです。カウンタがディレイモードにプログラムされると，MFPは8ビットカウンタのクロックにプリスケーラの出力を接続します。プリスケーラというのは，入力された周波数を固定比率で分周するものです。MFPは，プリスケーラの分周比を1/4，1/10，1/16，1/50，1/64，1/100，1/200の中から選択できるようになっています。

X68000ではプリスケーラへの入力として4MHzのクロックを与えていますので，たとえば，プリスケーラの分周比として1/100を選ぶと，8ビットカウンタには4MHz/100=40kHzのクロックが与えられることになります。クロックが1回入るたびに8ビットカウンタの値は減っていき，値が$01になると，次のクロックパルスでタイマ割り込みを発生させ，さらにタイマ出力端子(TAO/TBO/TCO/TDO)の状態を反転させます。8ビットカウンタにはタイマデータレジスタの値が自動的に再ロードされ，ふたたびカウントが始まります。したがって，最終的な分周比は，プリスケーラとタイマデータレジスタにセットした分周比の積になり

●図……7　MFPのタイマの各動作モード

ます。

たとえば，プリスケーラとして1/100を選び，タイマデータレジスタに400をセットすると，8ビットカウンタの出力は4 MHz/(400×100)=100 Hzとなり，10 msおきに割り込みが発生することになります。タイマ出力端子はこの周期で反転するわけですから，出てくる周波数はさらにこの半分の50 Hzとなります。

タイマAとタイマBには制御入力としてTAIとTBIがありますが，このモードでは使用さ

れません。

❻・❶ 2 パルス幅測定モード

パルス幅測定モードは，TAI/TBI 入力が指定されたレベルである期間だけタイマが動くようにすることで，入力された信号のパルス幅（Hレベルないし Lレベルが続いた時間）の測定が行えるようにしたモードす。このモードは，タイマAとタイマBだけで利用可能です。X 68000 では TBI 端子はLレベルに固定されてしまっていますので，実際にこのモードが利用できるのはタイマAだけになります。

パルス幅測定モードでは，タイマのスタート/ストップを TAI，TBI 入力で行い，タイマをストップさせたとき，すなわち，測定の完了時に CPU に割り込みをかけることができます。'H'と'L'のいずれのレベルでカウンタスタートとするかは，AER の GPIP 4，GPIP 3 の設定によって決まり，発生する割り込みはタイマAが GPIP 4，タイマBが GPIP 3 の割り込みになります。つまり，タイマAは GPIP 4 の割り込み機構を，タイマBは GPIP 3 の割り込み機構を乗っ取るようなかたちになるわけです。このため，タイマAをパルス幅測定モードにすると GPIP 4 の変化による割り込み発生が，タイマBをパルス幅測定モードにすると GPIP 3 の変化による割り込み発生が行えなくなります。もちろん，この場合でも，GPIP レジスタで GPIP 端子の状態の読み出し/設定（X 68000 では GPIP は読み出し専用ですが）は行えますから，たんなる I/O として利用することは可能です。

AER で'1'が設定されていると，TAI/TBI 入力が'H'レベルでタイマがスタートし，'L'レベルになるとストップするとともに CPU に割り込みが入ります（通常，GPIP 用として使っている場合，AER が'1'になっていると，'L'から'H'への変化で割り込み発生となりますが，パルス幅測定モードのときには，'1'にすると，'H'から'L'への変化で割り込みとなりますので注意してください）。

また，パルス幅測定モードは，基本的にタイマスタート/ストップ制御が外部信号で行われるディレイモードと同等ですから，カウントが$01 になった次のカウントクロックでタイマの割り込みも発生します。このとき，タイマにはタイマデータレジスタの値が自動的に再ロードされ，タイマストップ制御が行われるまでカウントを続けます。

測定が終了し，再度パルス幅測定を行う場合，CPU はタイマデータレジスタに値を再書き込みしますが，このとき，制御入力（TAI/TBI）がアクティブ（AER が'1'なら'H'レベル，'0'なら'L'レベル）になっていないことを確認してください。アクティブなときに書き込みを行うと，カウンタに正しい値がロードされない場合があります。

❻·❶3 イベントカウントモード

このモードも，タイマAとタイマBだけが使用可能です。イベントカウントモードは，TAIやTBIの入力をクロックとして8ビットカウンタを動作させるモードです（当然のことながら，プリスケーラは使用されなくなります)。入力のどちら方向の変化でカウントを行うかは，パルス幅測定モードと同様に，AERのGPIP 4/GPIP 3で行います。

カウンタの値が$01になった後にカウントパルスが発生すると，CPUに対して割り込み(タイマA/タイマBの割り込み)を発生するとともに，タイマカウントレジスタの値が自動的に再ロードされます。

X 68000ではTAI入力にV-DISP信号が接続されており，Human 68 KはタイマAをイベントカウントモードで使用しています。

❻·2 タイマ関連のレジスタ

タイマ制御を行うためのレジスタは，タイマの動作モードを設定するタイマコントロールレジスタと，8ビットカウンタの値のリード/ライトを行うためのタイマデータレジスタの2種類に分類できます。このうち，タイマCとタイマDはディレイモードでしか動作できないこともあって，コントロールレジスタは1本のレジスタに圧縮してしまっています。

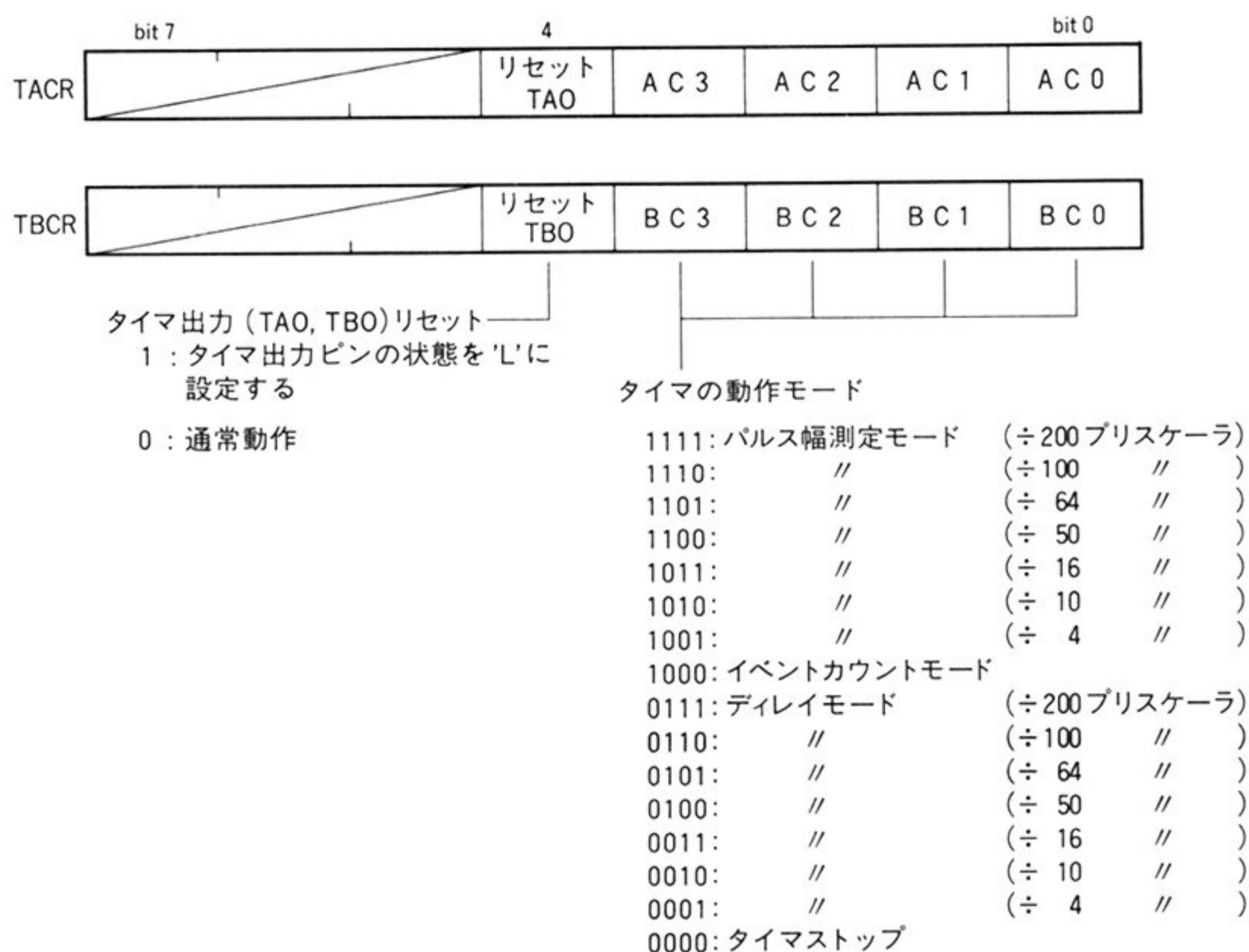
❻·❷1 タイマA/タイマBコントロールレジスタ

タイマAとタイマBのコントロールレジスタのビット配置を図8に示します。

下位4ビットは，それぞれのタイマの動作モードを指定するものです。'0000'のときにはタイマストップとなり，タイマ動作が禁止され，'1000'のときにはイベントカウントモードとなります。下位3ビットが'000'以外のときは，ビット3が'0'だとディレイモードが，'1'だとパルス幅測定モードが選択されます。

ディレイモードやパルス幅測定モードのときには，下位3ビットでプリスケーラの分周比を選択します。

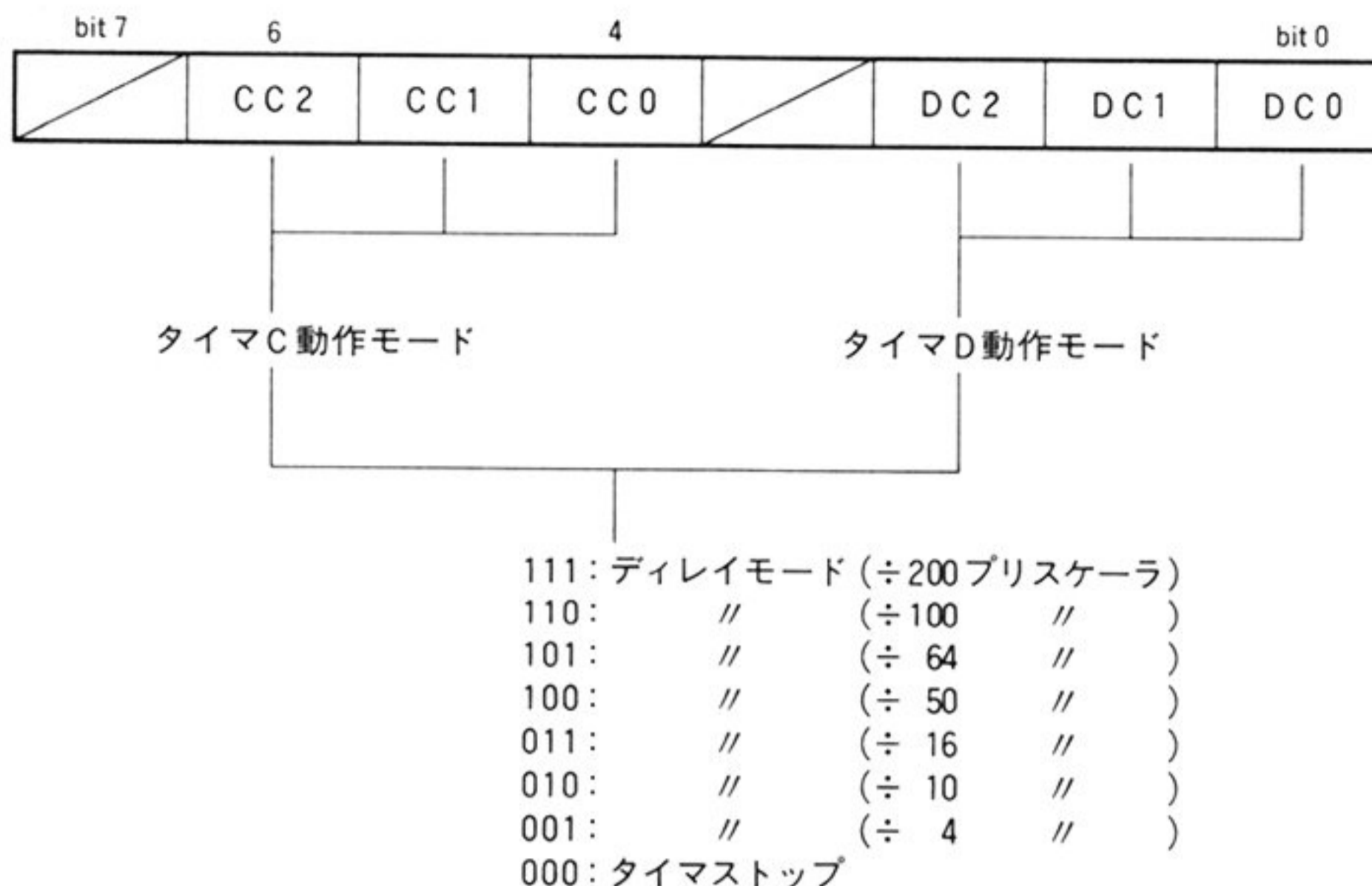
ビット4は，タイマ出力端子であるTAO/TBOの出力を強制的にクリアするためのものです。このビットを'1'にして書き込むと，タイマ出力端子の状態が強制的に'L'レベルになります。この機能によるクリアはCPUによる書き込み動作の期間だけ有効で，クリア後，次に8ビット

●図……8　TACR, TBCR($E88019, $E8801B)

カウンタからのカウントアップパルスがくれば，通常動作どおり出力は反転されます。タイマ出力が'L'の状態から動作開始させたいようなときのためにあると考えればよいでしょう。

❻・❷ 2　タイマC＆Dコントロールレジスタ

タイマCとタイマDのコントロールを行うレジスタのビット配置を92ページの図9に示します。タイマDの動作モードの選択をビット0〜ビット2で，タイマCの制御をビット4〜ビット6で行います。

この3ビットがすべて'0'のときには，タイマ動作が禁止されます。それ以外のときにはタイマCやタイマDはディレイモードで動作し，3ビットでプリスケーラの分周比の選択を行います。この設定は，タイマA/タイマBコントロールレジスタのモード設定の最上位ビットがつねに'0'であるとした場合と同じになります。

●図……9　TCDCR（タイマC＆Dコントロールレジスタ）

❻・❷ 3　タイマデータレジスタ

それぞれのタイマごとに1本ずつ，タイマの値のリード/ライトを行うためのタイマデータレジスタが用意されています。タイマはカウントパルスが入るたびに減少していき，$01になると，次のパルスでタイマデータレジスタに設定した値が自動的に再ロードされます。

●7　USART（シリアルポート）

MFP内蔵のUSART（Universal Synchronous/Asynchronous Receiver/Transmitter）は，全二重の同期通信/非同期通信の両方をサポートしている汎用のシリアルインタフェースです。X 68000ではキーボードと接続するように決められているため，キーボードの伝送モード（非同期，2400 bps，スタートビット1ビット，データ8ビット，パリティなし，ストップビット1ビット）にあわせることになります。また，X 68000では，USARTの伝送クロックはタイマBから得るようにしており，外部データに同期させるようなことはできないため，クロックモードも1/16以外は選択できません。

このように，X 68000 ではモード選択の余地はほとんどありませんが，一応どのような働きをするものなのかを知っておいたほうがよいと考え，MFP の USART の持つ機能を一通り説明しておくことにします。

❼·1 SCR(SYNCキャラクタレジスタ)

同期転送モード時，USART は SCR に設定されたデータが受信されるまで待ち続けます。また送信時には，送信データが書き込まれず，アンダーラン状態になると，自動的に SCR に設定されたキャラクタが送信されます。SCR への設定は，UCR の WL ビットで設定したデータ長（パリティが有効のときにはデータ長＋1）以上のビットは無効となり，'0'として扱われるため，SCR への設定は，必ず UCR の WL ビットを設定した後で行わなくてはなりません。

また，データ長が 8 ビットのとき以外は，USART はパリティを自動的には付加しませんので，ユーザ側で SYNC キャラクタにパリティを付加したデータを SCR に設定しなくてはなりません。

X 68000 では USART を非同期モードで使用しますので，SCR は無視してかまいません。

❼·2 UCR(USARTコントロールレジスタ)

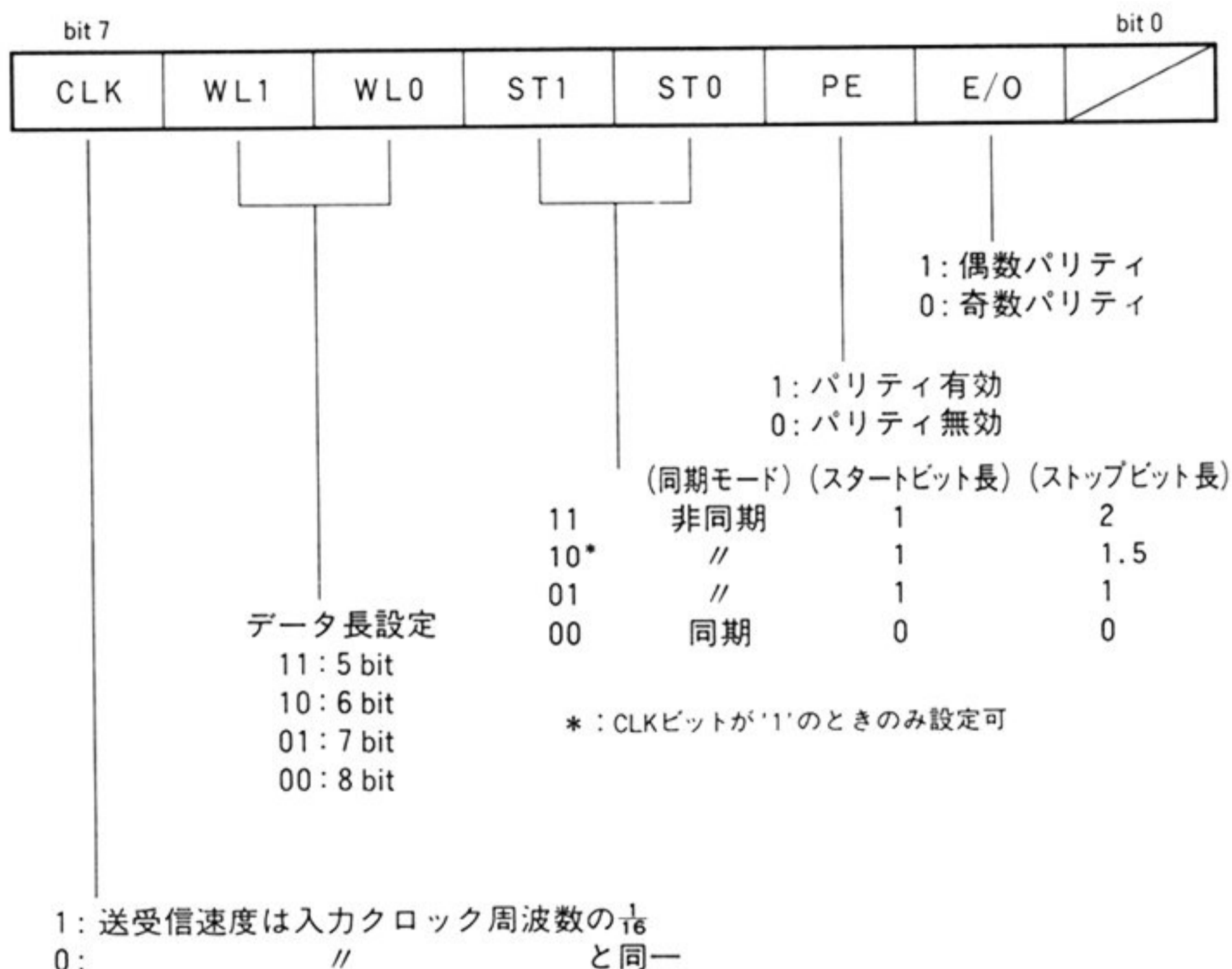
USART の動作モードを決めるレジスタです。UCR のビット配置を 94 ページの図 10 に示します。

❼·❷1 CLK

送受信速度を入力クロック周波数と同一にするか，送受信速度を入力クロック周波数の 1/16 にするかを決めます。'1'に設定すると 1/16，'0'に設定すると同一となります。

1/16 モードのときには，USART は入力されたデータからスタートビットを見つけ，自動的にデータビットの中心をサンプリングしながらデータを取り込みます。パソコン通信などで使われるモデムとパソコン本体の通信などは，このモードで行われています。X 68000 でも，キーボードとの通信はこのモードで使用します。入力クロックはタイマBの出力クロックですから，タイマBの出力周波数は 2400 (bps)×16 = 38400 Hz になるようにします。

送受信クロックが入力クロックと同一の場合，USART はクロックに同期して無条件にデー

●図……10　UCR（USART コントロールレジスタ）$E88029

タを取り込むため，データとクロックが完全に同期していないとデータが化けてしまいます。このため，このモードを選択したときにはデータとともにクロックも接続しておくか，受信されたデータから同期したクロックを生成するような外部回路が必要になります。X 68000では，クロックはMFPのタイマBに接続されていますので，このモードは選択できません。

❼·❷2　WL

1キャラクタのデータ長を設定します。'00'だと8ビット，'01'で7ビット，'10'だと6ビット，'00' のときには5ビットとなります。X 68000では，キーボードのデータ長が8ビットですから，'00'を設定することになります。

❼·❷3　ST1, ST0

スタートビット，ストップビットの長さ，同期/非同期モードの選択を行います。'00'を設定すると同期モードとなり，スタートビット，ストップビットとも0になります。'00'以外の場合

は非同期モードとなります。このうち，設定値'10'，すなわちスタートビット１ビット，ストップビット1.5ビットのモードは，CLKが１のとき（1/16モードのとき）だけ設定可能です。X68000のキーボードは，スタートビット，ストップビットとも１ビットですから，このビットは'01'を設定することになります。

❼・❷4 PE

パリティを有効とするか，無効とするかを選択します。'1'を設定するとパリティが有効となります。受信時にはパリティチェックが行われ，送信時にはデータの後にパリティビットが自動的に付加されます。ただし，８ビット以下のSYNCキャラクタに対しては，PEが'1'になっていても，パリティビットは付加されません（データには必ず付加されます）ので注意してください。

❼・❷5 E/O

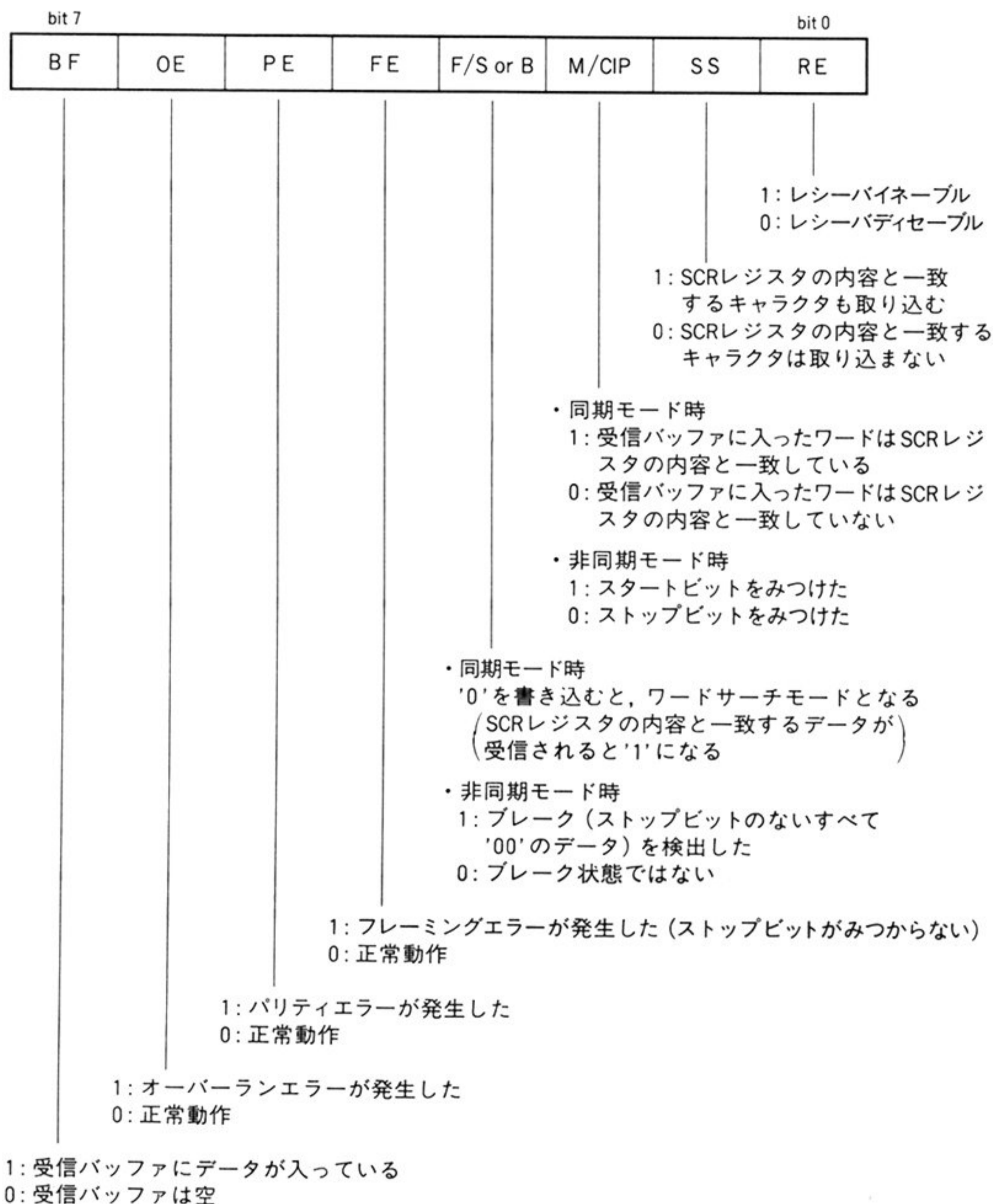
パリティを偶数パリティとするか，奇数パリティとするかを選択します。'1'のときは偶数パリティ，'0'のときには奇数パリティになります。

❼・3 RSR(レシーバステータスレジスタ)

RSRは，受信ステータスの読み出しや，レシーバのイネーブル/ディセーブルの制御などを行うレジスタです。RSRのビット配置を96ページの図11に示します。

❼・❸1 BF

BF(バッファフル)ビットは，受信バッファにデータが入っているか否かを示すビットです。受信バッファにデータが入っていると'1'になり，UDR(USARTデータレジスタ)をCPUが読み出し，バッファのデータを引き取ると'0'になります。.

●図……11　RSR（レシーバステータスレジスタ）$E8801B

❼·❸ 2　OE

OE(オーバーランエラー) は，受信バッファに入ったデータがCPUによって引き取られないまま，次のデータが入ってきてしまった場合に発生します。新しく入ってきたデータは捨て

られます。OEビットは，オーバーランエラー発生後，受信バッファに入っているデータが読み出された時点で'1'になり，RSRレジスタを読み出すと'0'になります。

❼·❸3 PE

PE（パリティエラー）は，受信されたデータから計算したパリティと，受信されたパリティが一致しないと発生します。エラーが発生すると'1'に，エラーのないデータが受信されると'0'になります。

❼·❸4 FE

FE（フレーミングエラー）は非同期モードのときだけ有効です。$00以外のデータを受信した後，ストップビットが見つからないとフレーミングエラーが発生し，FEビットが'1'になります。正常なデータが受信できると'0'に復帰します。

❼·❸5 F/S or B

F/S or B（ファウンド/サーチまたはブレーク）ビットは，同期モード，非同期モードの別によって機能が変わります。

同期モード時には，'0'を書き込むとワードサーチモードになり，SCRレジスタの内容と一致するデータが受け取られるまで待ちます。SYNCキャラクタと同じデータが受信されると'1'になり，CPUに知らせるため，受信エラー割り込みを発生します。

非同期モードのときには，データラインがブレーク状態になったことを検出したときに'1'となるステータスビットになります。ブレーク状態は，データラインが'0'のままになっている状態で，ストップビットの見つからない$00のデータと考えることができます($00以外のときにストップビットが見つからないとフレーミングエラーになります)。

F/S or Bビットは，$00以外のデータが受け取られ，RSRが読み出されると，'0'に復帰します。

❼・❸ 6 M/CIP

M/CIP (マッチ/文字処理中) ビットも，同期モード，非同期モードの別によって機能が変わります。

同期モードの場合，SYNC キャラクタと同じデータが受信バッファに入ったときに'1'になり，一致しないキャラクタが受信バッファに入ると'0'に復帰します。

非同期モードの場合，スタートビットが見つかると'1'になり，ストップビットが見つかると'0' に復帰するようになります。

❼・❸ 7 SS

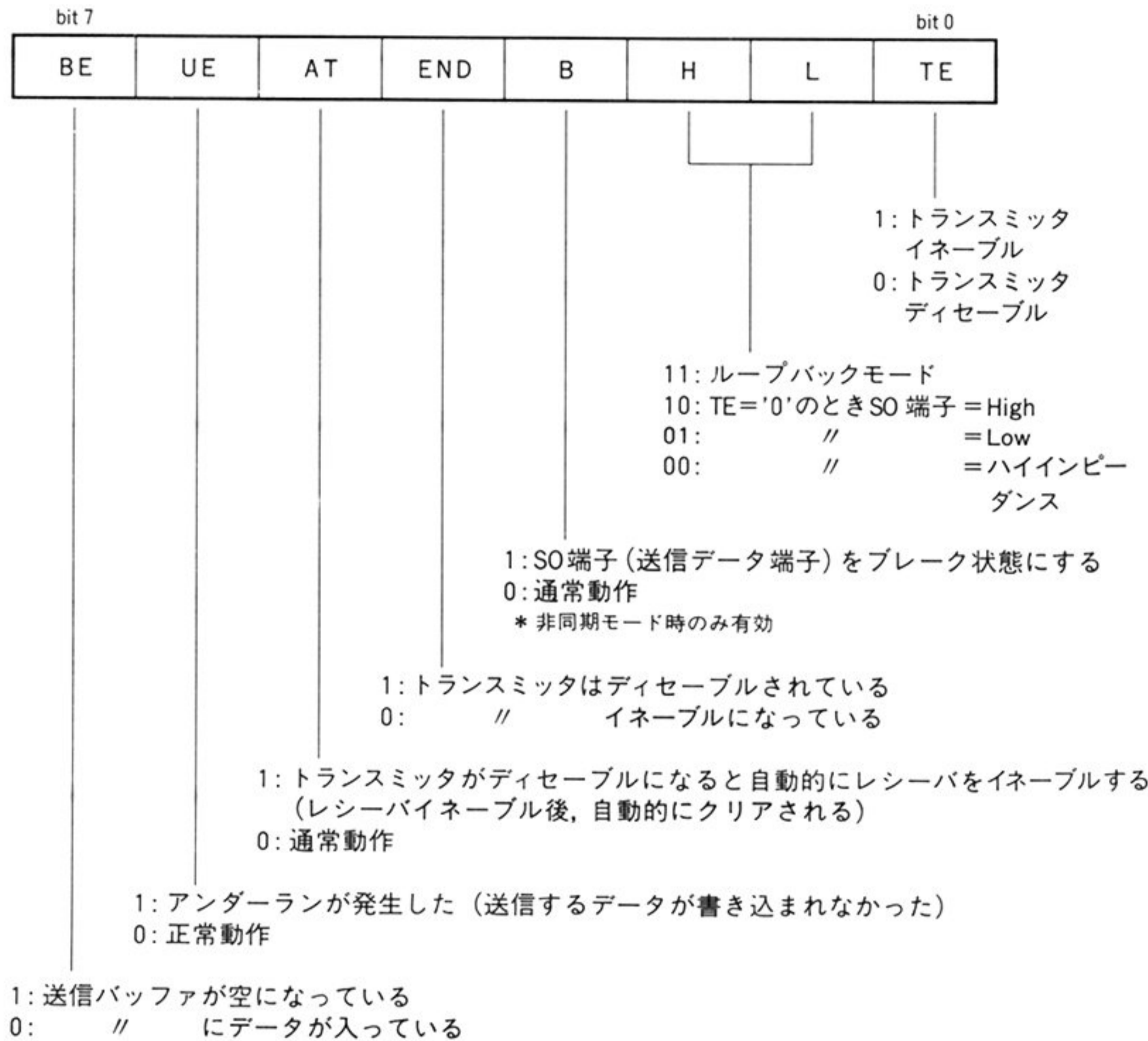
SS(シンクロナスストップ)ビットは，SYNC キャラクタを受信するか否かを決めるビットです。SS ビットが'0'になっていると，SYNC キャラクタと一致するデータは受信バッファには入らず，当然，バッファフルにもなりません。

❼・❸ 8 RE

RE (レシーバイネーブル) ビットは受信動作のイネーブル/ディセーブルの制御を行います。RE ビットを'0'にすると，受信動作は中止され，RSR の各ステータスビットは'0'になります。'1'になると受信動作はイネーブルとなりますが，このとき，受信クロックが供給されていなければなりません。

❼・4 TSR(トランスミッタステータスレジスタ)

TSR のビット配置を図 12 に示します。TSR は，送信状態や送信動作モードの設定を行うレジスタです。

●図……12 TSR（トランスミッタステータスレジスタ）

❼・❹1 BE

BE（バッファエンプティ）ビットは，送信バッファが空になっていることを示すビットです。送信バッファが空になると，BEビットは'1'になり，UDR（USARTデータレジスタ）にデータが書き込まれると，BEビットは'0'に復帰します。

❼・❹2 UE

UE（アンダーランエラー）ビットは，送信バッファにデータが書き込まれないまま，最後のデータが送信し終わってしまった場合に発生します。TEビットによって送信をディセーブル

したり，TSR レジスタを読み出すと，UE ビットはクリアされます。

❼·❹3 AT

AT(オートターンアラウンド)ビットが'1'になっていると，最後のデータの送信が終わった時点で自動的にレシーバがイネーブルになります。送信が終了した時点で，このビットは自動的に'0'になります。

❼·❹4 END

データが送信されているときにトランスミッタをディセーブルする(TE を'0'にする)と，データの送信が終了した時点で END(送信終了)ビットが'1'になります。トランスミッタがイネーブルされると，END ビットは'0'に復帰します。

❼·❹5 B

B (ブレーク) ビットは，非同期モードのときだけ有効です。非同期モードのときに B ビットを'1'にすると，現在送信中のデータが送信し終わった後で送信データラインをブレーク状態にします。B ビットを'0'にすると，ブレーク状態は中止され，通常状態に復帰します。このビットが'1'になっている間，BE ビットが'1'になることはありません。

❼·❹6 H, L

H，L(High/Low)ビットは，トランスミッタをディセーブルにしたときの送信データラインの状態を決めるものです。'00'のときはハイインピーダンス，'01'のときは'L'レベル，'10'のときは'H'レベルになります。'11'のときは少し特殊で，ループバックモードという一種の自己診断モードに入ります。このモードのとき，受信データラインと受信クロックラインが MFP 内部で送信データラインと送信クロックラインに接続され，送信したデータがそのまま受信される折り返し試験が行えます。通常，このビットには'10'を設定しておくとよいでしょう。

❼·❹7 TE

TE（トランスミッタイネーブル）ビットは送信動作の許可/禁止を制御します。TEビットが'1'になっていると，送信動作がイネーブルとなり，データの送信が行えるようになります。

❼·5 UDR(USARTデータレジスタ)

UDRはデータの受け渡しを行うレジスタです。ここに書き込まれたデータは，送信ラインを使って送出され，受信されたデータはこのレジスタを通してCPUに受け取られます。

8 MFPの初期設定

MFPの各レジスタの設定値の一覧を102ページの図13に示します。'1'あるいは'0'となっているビットは，その設定値で固定であることを，Pは設定を変更できるビットを，Xは読み出し専用のビットや，書き込み時'1'と'0'のいずれであってもかまわないビットを示しています。

システム設定値のデータは，Human 68 Kを起動した後で読み出した設定値です。タイマデータレジスタは変化しているので，10万回ほど連続して読み出したときの最大値を表に記入しておきました。

●図……13　MFPの設定値

アドレス	bit7							bit0	システム設定値	レジスタ名
$E88001	X	X	X	X	X	X	X	X	——	GPIPデータレジスタ
$E88003	0	0	X	P	X	1	X	X	$06	アクティブエッジレジスタ
$E88005	0	0	0	0	0	0	0	0	$00	データディレクションレジスタ
$E88007	P	P	P	1	1	P	P	0	$18	割り込みイネーブルレジスタA
$E88009	P	0	P	P	0	1	0	0	$3E	割り込みイネーブルレジスタB
$E8800B	X	X	X	X	X	X	X	X	——	割り込みペンディングレジスタA
$E8800D	X	X	X	X	X	X	X	X	——	割り込みペンディングレジスタB
$E8800F	X	X	X	X	X	X	X	X	——	割り込みインサービスレジスタA
$E88011	X	X	X	X	X	X	X	X	——	割り込みインサービスレジスタB
$E88013	P	P	P	1	1	P	P	0	$18	割り込みマスクレジスタA
$E88015	P	0	P	P	0	1	0	0	$3E	割り込みマスクレジスタB
$E88017	P	P	P	P	P	X	X	X	$40	ベクタレジスタ
$E88019	0	0	0	0	1	0	0	0	$08	タイマAコントロールレジスタ
$E8801B	0	0	0	0	0	0	0	1	$01	タイマBコントロールレジスタ
$E8801D	0	P	P	P	0	P	P	P	$77	タイマC＆Dコントロールレジスタ
$E8801F	P	P	P	P	P	P	P	P	$01	タイマAデータレジスタ
$E88021	0	0	0	0	1	1	0	1	$0D	タイマBデータレジスタ
$E88023	P	P	P	P	P	P	P	P	$C8	タイマCデータレジスタ
$E88025	P	P	P	P	P	P	P	P	$14	タイマDデータレジスタ
$E88027	0	0	0	0	0	0	0	0	$00	SYNCキャラクタレジスタ
$E88029	1	0	0	0	1	0	0	X	$88	USARTコントロールレジスタ
$E8802B	X	X	X	X	X	X	0	P	$01	レシーバステータスレジスタ
$E8802D	X	X	P	X	P	1	0	P	$81	トランスミッタステータスレジスタ
$E8802F	X	X	X	X	X	X	X	X	——	USARTデータレジスタ

X…読み出し専用/任意のデータで可　P…必要に応じて設定変更可

数値演算プロセッサ

数値演算プロセッサは浮動小数点演算を実行するLSIで，レイトレーシングなど実数演算の多い用途で処理速度を大幅に向上することができます。ここでは，数値演算プロセッサの具体的な使用方法などについて説明します。

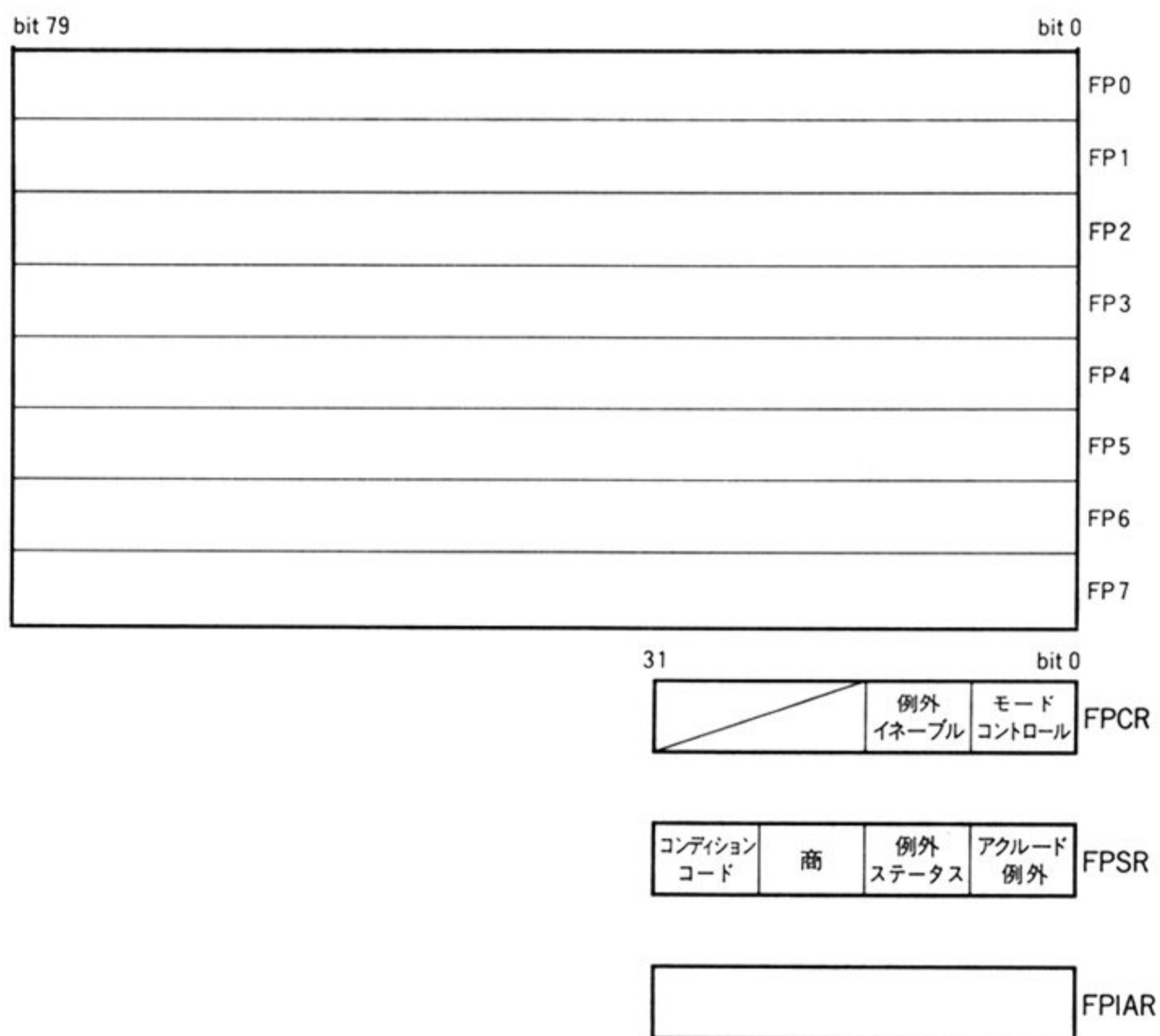
1 概要

数値演算プロセッサは数値演算，とくにCPUが苦手とする浮動小数点演算を高速に実行するLSIです。X 68000では，数値演算プロセッサとして68000ファミリーのMC 68881をオプションで搭載できるようにしています。搭載する形態は，XVI以前の機種では拡張ボード(CZ-6 BP 1)，XVI以降は本体内部の専用ソケットに挿入，と異なっていますが，ソフトウェアから見た場合にはまったく同じものとなっています。

68881は，もともと68020と直結し，コプロセッサとして使うのが本来の姿なのですが，68000などの他の一般的なCPUと接続することもできるようになっています。X 68000では，68881の，この機能を利用して，周辺I/Oデバイスとしてアクセスするようにしています。68020の場合には，CPUがコプロセッサ専用の命令を解釈し，68881とのこまごまとしたやりとりをすべて自動的にこなしてくれるのですが，X 68000のような使い方の場合には，このあたりの操作をすべてソフトウェアで行わなくてはならないため，扱いが少々面倒になっています。この章では，まず，68881内部の演算処理機能や演算命令の説明などを行い，最後に68881との細かなやりとりの方法を説明していくことにします。

●2 68881の内部レジスタ

68881の内部レジスタの一覧を図1に示します。これらのレジスタへのアクセスは，あくまでも演算命令やデータ転送命令などを利用して行われるものであり，CPUから見て，あるアドレスに直接配置されるものではありませんので注意してください。

●図……1　68881の内部レジスタ

❷·1 FPn

FP 0 から FP 7 の 8 本の 80 ビット長のレジスタは浮動小数点データレジスタです。68881 の演算処理などは，これらのレジスタを使用して行います。8 本のレジスタは，まったく同等のものであり，あるレジスタだけが特殊なものとして扱われるようなことはありません。ちょうど CPU のデータレジスタ（D 0～D 7）に相当するようなものと考えればよいでしょう。

❷·2 FPCR, FPSR, FPIAR

FPCR は，68881 が発生する例外的なイネーブル/ディセーブルの制御，演算結果の丸め処理の指定などを行うものです。FRCR の PREC ビット（図 2）によって，丸め精度を単精度や倍精度に変更できる機能は，あくまでも拡張精度での演算が行えない他の計算機との互換性を維持するためのものであり，演算速度も，拡張精度のときよりもかなり落ちてしまいますので，通常拡張精度以外を指定する必要はないでしょう。

FPSR は，演算エラーやオーバフローなどが起こってしまったときに，状況の解析や後始末を行う際に有効なステータスや除算命令の商データなどが格納されます。

FPSR のコンディションコードバイトは演算命令の終わりでセットされるものです。107 ページの図 3 に示した各条件が成立すると'1'になります。

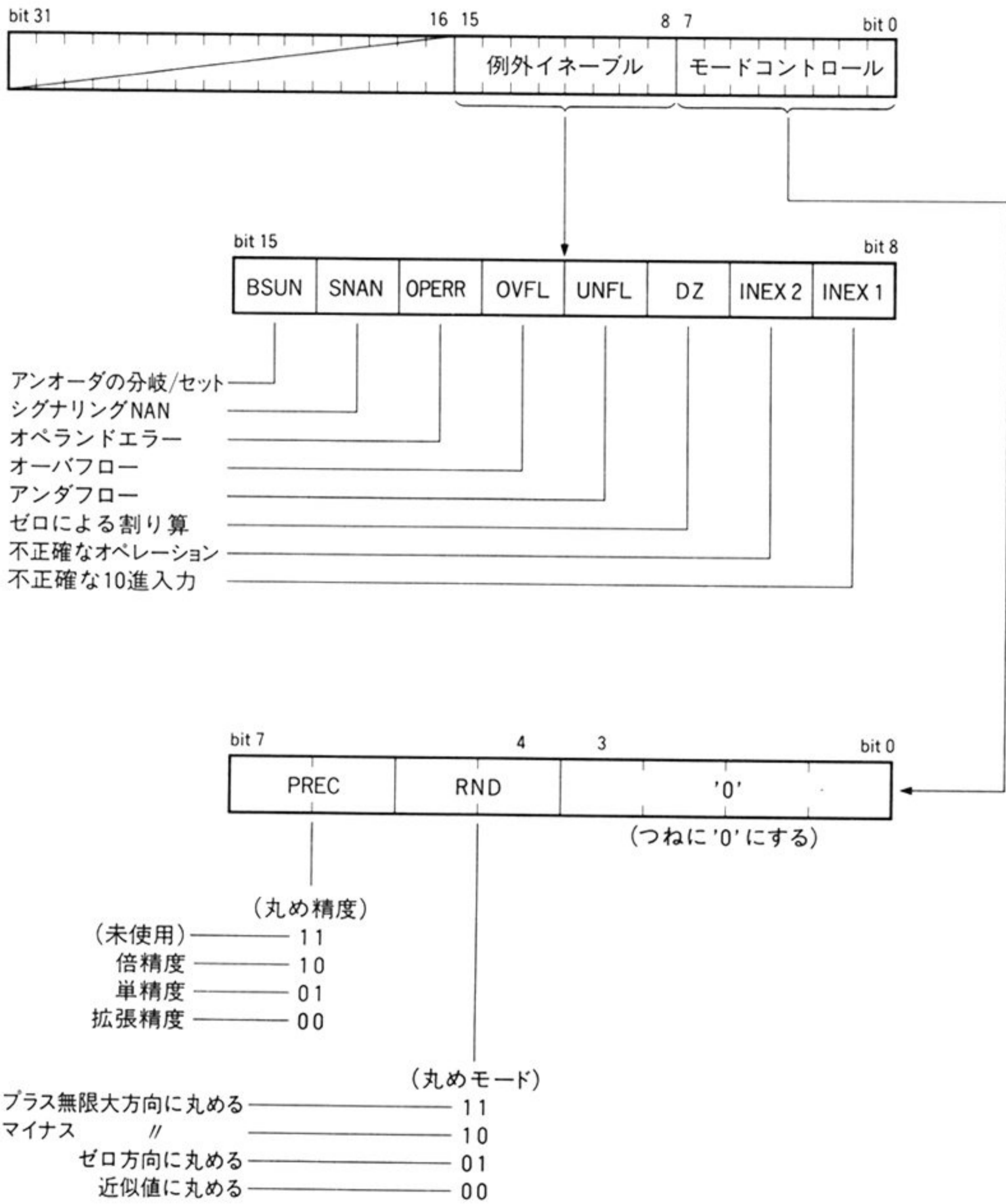
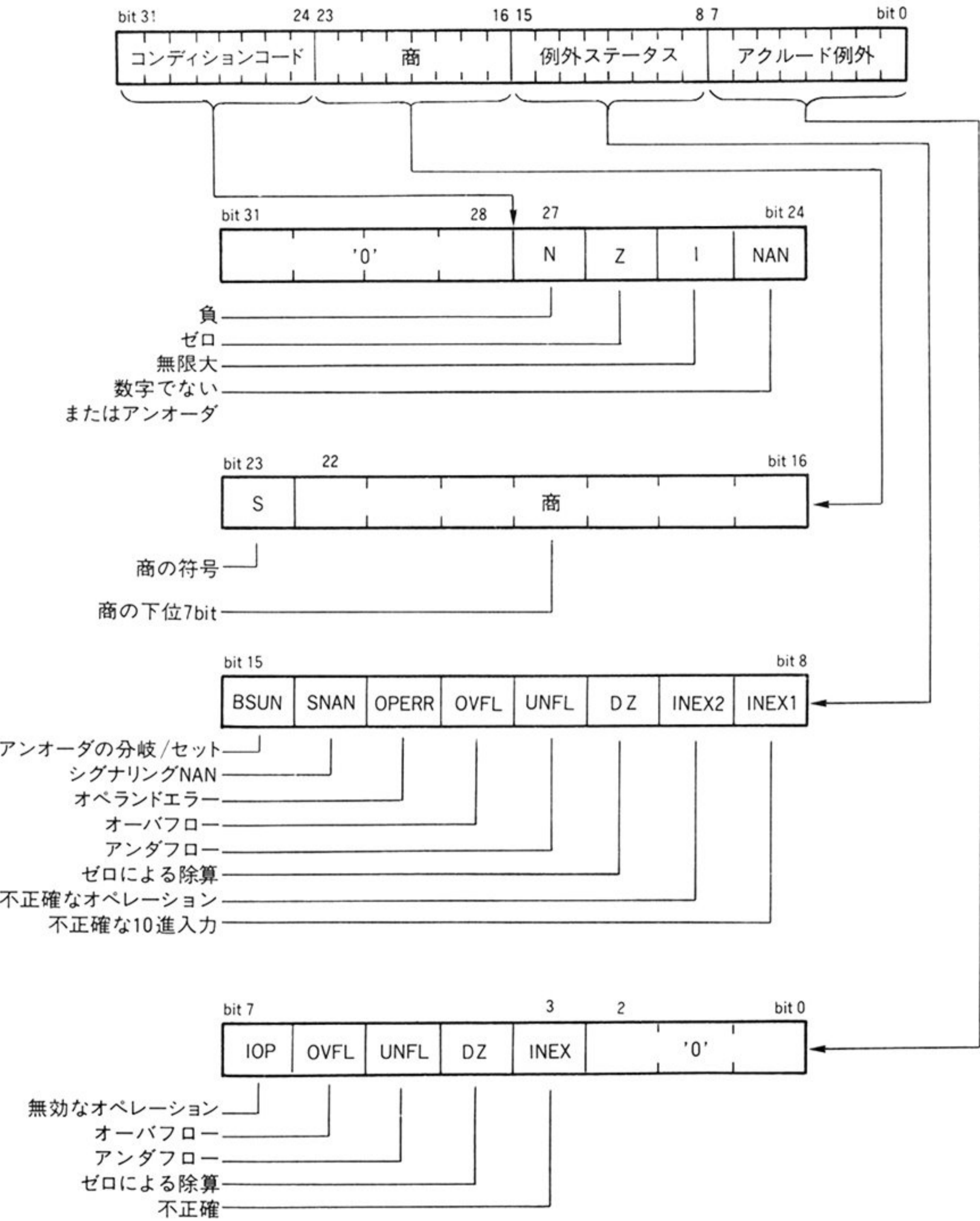
商バイトは，モジュロ（FMOD）命令や IEEE 剰余（FREM）命令を実行したときにセットされます。

例外ステータスバイトは，最後に行われた浮動小数点演算やデータ転送で発生したエラーやオーバフローなどの例外状態を示すために使用されます。

アクルード例外バイトは，IEEE で規定されている 5 種類の例外ビットが入っています。この各ビットは例外ステータスバイトから生成されますが，例外ステータスバイトが演算のたびにセット/リセットされるのに対して，アクルード例外バイトは発生した条件が OR されていきます。これにより，一連の演算処理の前にアクルード例外バイトをクリアしておき，終了後に 0 のままになっているかどうかをチェックするだけで，一連の演算がすべて問題なく行われたかどうかを知ることができます。

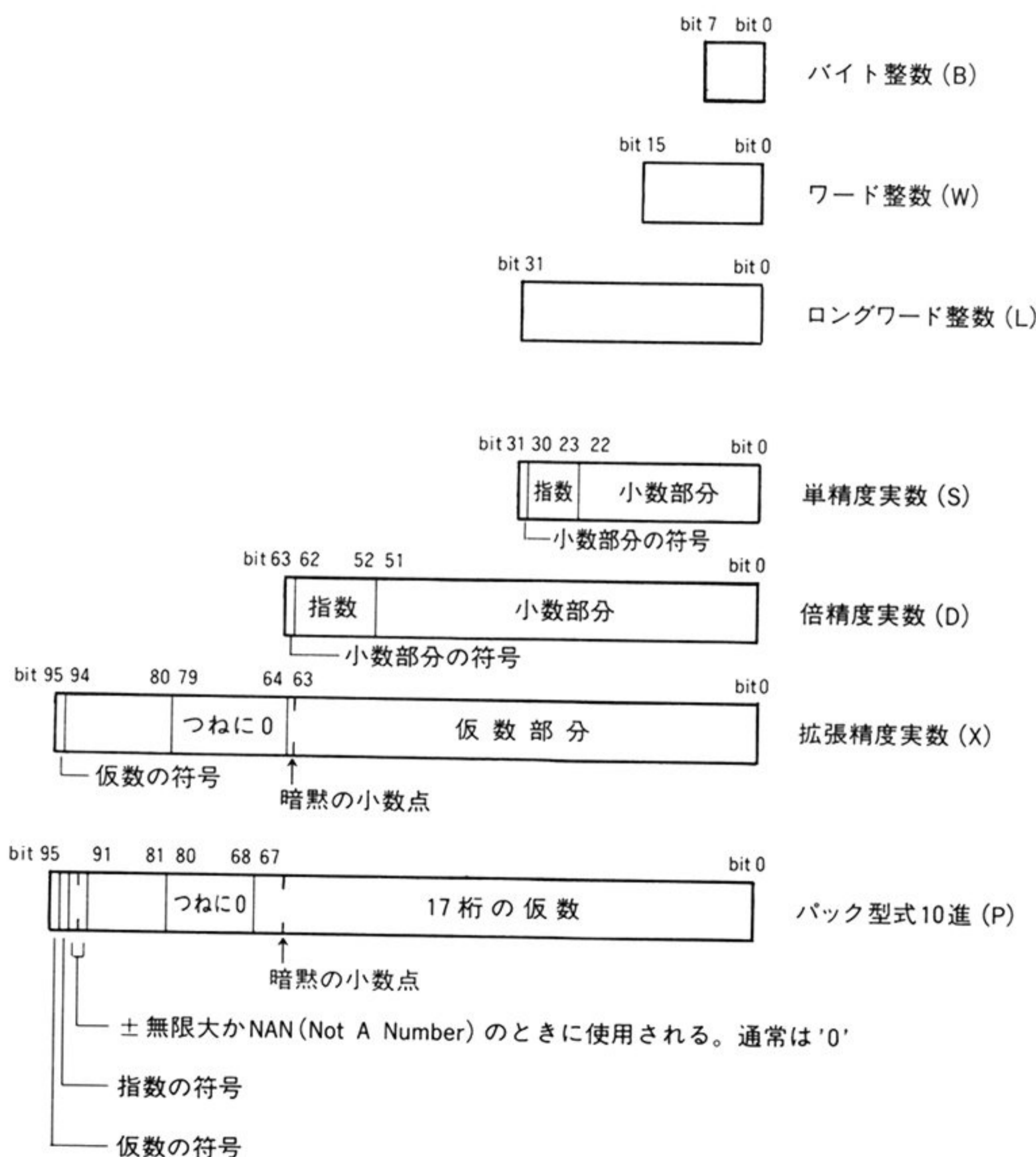
FPIAR は，実行された最後の浮動小数点命令のアドレスを保持するものです。このレジスタは 68020 と直結した場合に，割り込みによって中断された演算処理の実行を，割り込み処理の終了後に再開するために使用されるものです。X 68000 の場合のように 68881 を I/O デバイスとして接続したときにはあまり意味がないレジスタですが，アクセスすることは可能です。

FPCR と FPSR の詳細を図2と図3に示しますので参考にしてください。

●図……2　FPCR（コントロールレジスタ）

●図……3　FPSR（ステータスレジスタ）

3 68881が扱えるデータフォーマット

68881が外部とのやりとりで扱うことのできるデータフォーマットを図4に示します。

68881の内部演算自体はつねに拡張精度で行われており，また，浮動小数点データレジスタにはFPCRで指定した丸め精度でデータが格納されています。内部で保持している精度と指定

●図……4　68881が扱うことができるデータのフォーマット

された型が異なる場合，68881は自動的に型変換を行います。つまり，外部から転送されたデータは必ず内部で拡張精度に変換され，外部に転送するときには拡張精度から指定された型に変換された後，転送動作が行われるわけです。

各フォーマットの横のカッコの中に書いてあるアルファベットは，その型の略称としてメーカであるモトローラが推奨しているものです。68000のアセンブラで'MOVE.B'のように型指定を行いますが，それと同じようなもので，たとえば，単精度実数の転送では，'FMOVE.S'のように記述します。

❸·1 実数データのフォーマット

バイト，ワード，ロングワードの各整数型は，すべて68000 CPUで扱われる整数データと同じですので，とくに説明はいらないでしょう。

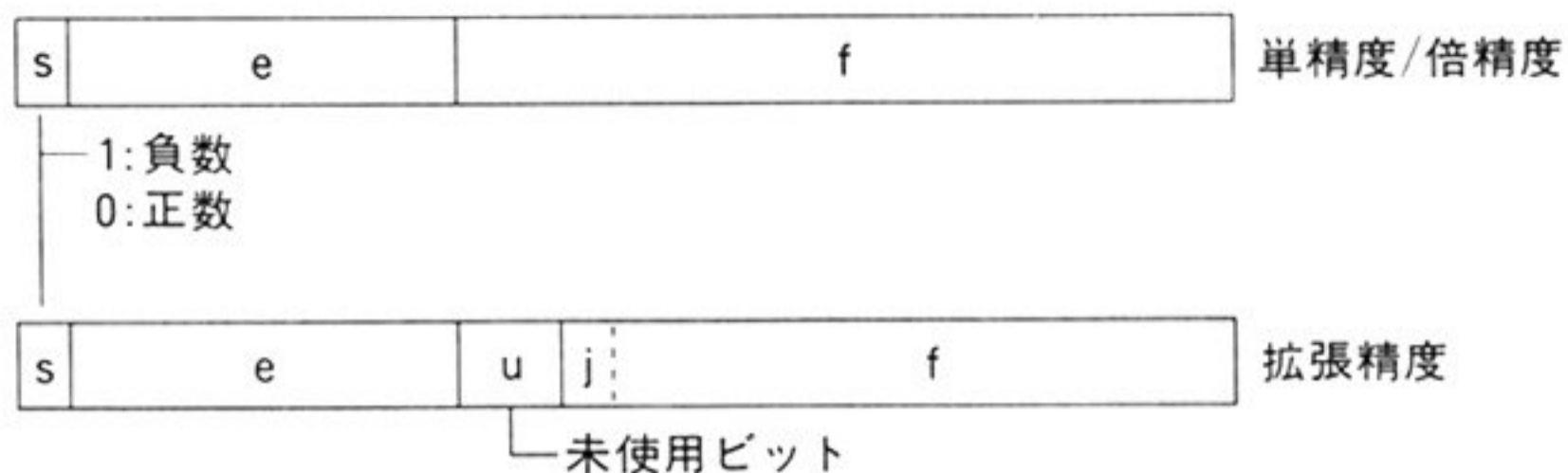
単精度，倍精度，拡張精度の各実数フォーマットはすべてIEEE規格に準じています。ただし，拡張精度のうち，ビット64からビット79までの16ビットはつねにゼロであるため，68881内部では省略され，80ビットデータとなっています。

10進数で実数を表現するとき，7.2×10の3乗といったように，整数部分を1桁として表現しますが，IEEEによる2進数の実数表現も，これと同じように整数部分を1桁にした仮数部と指数部に分けて表します。10進数の場合には，整数部分には1から9までの数値がきますが，2進数では1にしかなりえません。小数部分を仮りにf，指数をeで表せば，$1.f\times2^{e}$という表現になるわけです。

単精度実数と倍精度実数では，この無駄な整数部分を省略し，データ中には小数部分だけを格納しています。拡張精度実数では，64ビットの仮数部の最上位ビットが整数部分で，上位から2桁目以降が小数部分として扱われます。

指数部分は正，負いずれの場合も存在しますので，表せるデータの半分あたりの値にオフセットをかけています。たとえば，単精度実数なら，指数部は8ビットありますので，$7F(127)だけ足した値が格納されます。指数部が2^{0}なら指数データは$7F，$2^{1}$なら$80，2^{-3}なら$7Cとなるわけです。

各実数フォーマットにおける実数の表現を110ページの図5にまとめておきましたので参考にしてください。図中，正規化数とあるのが通常の浮動小数点データの表現です。非正規化数というのは，値の絶対値があまりにも小さくなり，アンダフローを起こす限界のときの値で，指数部分が0，仮数部の整数データが0となっているデータの扱いを示しています（仮数部の小数データもすべて0になっていると，ゼロを示すことになります）。通常，単精度や倍精度の場合，仮数部分の整数はつねに1として扱いますが，指数が0のときには，例外として整数部

●図……5　実数のフォーマットのまとめ

		単精度	倍精度	拡張精度
各フィールドのビット長	s	1	1	1
	e	8	11	15
	u	—	—	16
	j	—	—	1
	f	23	52	63
	計	32	64	96
正規化数の表現		$(-1)^s \times 1.f \times 2^{e-127}$	$(-1)^s \times 1.f \times 2^{e-1023}$	$(-1)^s \times j.f \times 2^{e-16383}$
非正規化数の表現		$(-1)^s \times 0.f \times 2^{-126}$	$(-1)^s \times 0.f \times 2^{-1022}$	$(-1)^s \times 0.f \times 2^{-16383}$
表現可能な数の絶対値（概数）	正規化数最大	3.4×10^{38}	1.8×10^{307}	6×10^{4931}
	〃　最小	1.2×10^{-38}	2.2×10^{-308}	8×10^{-4933}
	非正規化数最小	1.4×10^{-45}	4.9×10^{-324}	9×10^{-4952}

分が0であるという扱いで数値を表現しますので注意してください。

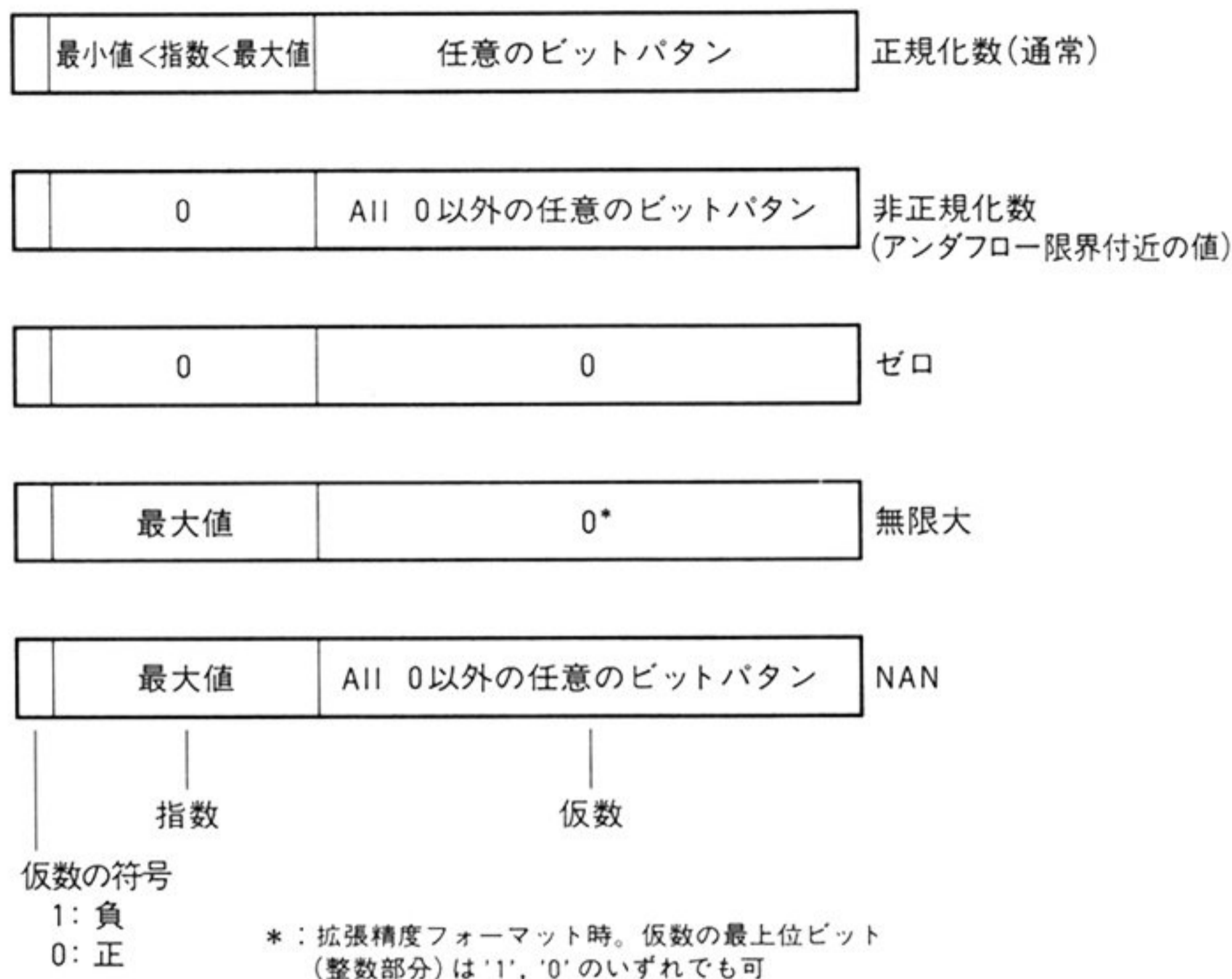
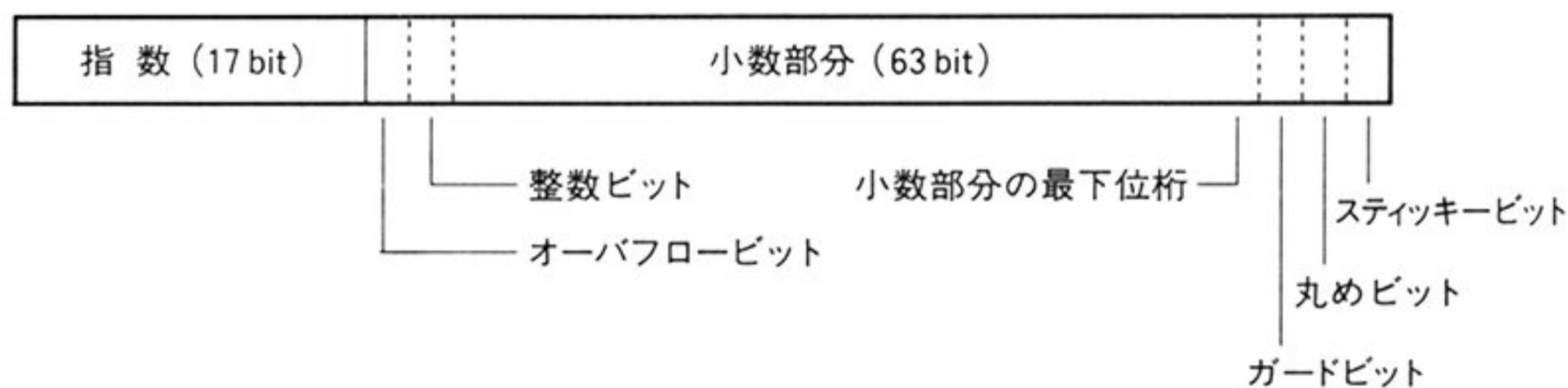
❸・2　特殊な実数データ

実数演算を行っていると，特殊な条件への配慮不足や，数学的には問題がなくても，68881が表現できるデータの範囲に限界があるために結果が0や無限大といったものになる場合があります。これらを通常の正規化数，先ほど説明した非正規化数とともに図6にまとめておきました。

最後のNANというのはNot A Numberの頭文字をとったもので，無限大÷無限大など，数学的に意味を持たない演算を行った場合に68881が演算結果として返すものです。

❸・3　68881内部のデータフォーマット

68881内部での演算処理途中の結果は図7のようなフォーマットとなっています。演算を繰り返したときの精度落ちを防ぐため，仮数部分は拡張精度の64ビットに対して67ビットとな

●図……6　特殊な実数データのフォーマット

●図……7　68881内部での演算途中のフォーマット

っており，また乗算命令実行時のオーバフローやアンダフロー検出などを容易にするため，指数が17ビット用意されています。

4　68881とのインタフェース

68881とX68000のコミュニケーションをとるためのレジスタ一覧を112ページの図8に

示します。これらのレジスタは，68020が数値演算プロセッサやメモリマネジメントユニットなどの各種のコプロセッサとコミュニケーションをとるために規定したCIR（コプロセッサインタフェースレジスタ）の規定にもとづいています（一部不要なレジスタは省略されています）。

68881が68020と直結された場合には，これらのレジスタとのやりとりはCPUである68020が自動的に行うため，プログラマがレジスタの存在を意識する必要はありませんが，X68000の場合には，68881をI/Oデバイスとして接続していますので，CPUになりかわってソフトウェアでこれらのレジスタをコントロールする必要があります。このため，演算のパフォーマンスはどうしても直結した場合よりも落ちますが，I/Oデバイスとしているため，複数の68881を同時にコントロールすることも可能となります。シャープ純正の数値演算プロセッサボードCZ-6BP1では，ピン設定によって2種類のアドレスを選択することができるようになっています。レジスタのアドレスは，標準設定では$E9E000～$E9E01F，ピン設定の変更で$E9E080～$E9E09Fとなります（Human 68 Kで使用される浮動小数点演算ドライバでは，このうち標準設定側しかサポートされていません）。

●図……8　CIR（コプロセッサインタフェースレジスタ）

ベースアドレス：$E9E000（標準）
　　　　　　　　$E9E080（2枚目）

(R)　：読み出し専門
(W)　：書き込み専用
(R/W)：読み書き可

❹·1 応答CIR

応答 CIR は，68881 が自分自身の動作状態やホスト CPU によるサービスの要求を示すために使用されます。応答 CIR はいつでも読み出すことができます。ホスト CPU は，このレジスタの値(プリミティブと呼びます)をチェックしながら動作することで，68881 と同期をとる（歩調をあわせる）ことができます。応答 CIR の内容の詳細は後で説明します。

❹·2 コントロールCIR

68020 のコプロセッサインタフェースの規定では，コントロール CIR は，ホスト CPU がコプロセッサに対して例外アクノリッジや命令の実行アボートを指示するために使用するものとなっています。68881 では，このレジスタへの書き込みをすべてアボート命令として受け取ります。このレジスタに書き込みが行われると，68881 は実行中の処理をただちに中止し，ペンディングされている例外（演算エラーなど）をすべてクリアした後，アイドル状態に復帰します。

68881 に例外が発生したような場合，ホスト CPU は，このレジスタに書き込み動作を行い，異常状態から回復させます。

❹·3 セーブCIR

ソフトウェアではアクセスされない 68881 の内部状態を読み出すために用意されているレジスタです。マルチタスク OS などでは，複数のタスクが 68881 を使用する可能性がありますが，タスクが切り替わったときに，68881 がまだ前のタスクが発行した演算命令を実行中であると，おかしなことになってしまいます。このような事態を避けるため，現在処理している状態をそのままメモリなどにセーブしておき，次にふたたび同じタスクに戻ってきたときに，その内容を回復して，中断された演算処理の続きをやらせる必要があります。このような目的で設けられているのが FSAVE と FRESTORE 命令で，セーブ CIR は FSAVE 命令の実行のために設けられているレジスタです。

このレジスタを読み出すと，68881 は現在の処理動作を中断し，動作状態ステータスを返します。ホスト CPU は，返されたデータを見て必要な分のデータを読み出します。

❹・4 リストアCIR

FRESTORE 命令を実行するためのレジスタです。ホスト CPU は，このレジスタにセーブ CIR を読み出したときに最初に返されたデータ（ステートフレーム）を書き込みます。このレジスタへの書き込みが行われると，68881 は与えられたステートフレームのフォーマットをチェックした後，リストア動作を開始します。ホスト CPU は，残るデータを 68881 に書き込み，中断されていた動作を再開させます。

フォーマットが不正であった場合，ホスト CPU はコントロール CIR への書き込みを行い，68881 をアイドル状態に復帰させます。

❹・5 オペレーションワードCIR

68881 は，このレジスタを使用しません。このレジスタへの書き込みは無視されます。

❹・6 コマンドCIR

ホスト CPU が 68881 に命令を書き込むために使用します。各種の演算命令やデータ転送命令は，すべてこのレジスタへの書き込みで開始されます。コマンドの詳細は，後で説明します。

❹・7 コンディションCIR

68020 と直結された場合，このレジスタは浮動小数点の条件付き命令（条件分岐命令など）を実行するときに使用します。X 68000 では，68881 は I/O デバイスとして接続されていますので，この CIR は条件チェック（等しい，大きい，小さいなど）を行うために使用できます。

❹・8 オペランドCIR

ホスト CPU と 68881 との間のデータ転送に使用されます。浮動小数点データの受け渡しなども，このレジスタを通じて行います。

④・9 レジスタ選択CIR

複数浮動小数点データレジスタ転送命令（FMOVEM 命令）を実行するとき，68881 からホスト CPU にレジスタマスクを渡すために使用します。ホスト CPU は渡されたデータの 1 の数をカウントすることで，転送するレジスタの数を知ることができます。

④・10 命令アドレスCIR

応答 CIR の PC ビットがセットされているときに，ホスト CPU が PC（プログラムカウンタ）の値を渡すために使用します。68881 が命令を実行しているときに，割り込みなどが発生する可能性がある場合，現在の PC の値を 68881 に渡します。X 68000 のように I/O デバイスとして接続した場合にはあまり利用する意味はないでしょう。ここへの書き込み要求は無視してもかまいません。

④・11 オペランドアドレスCIR

68881 は，この CIR を使用しません。アクセスはすべて無視されます。

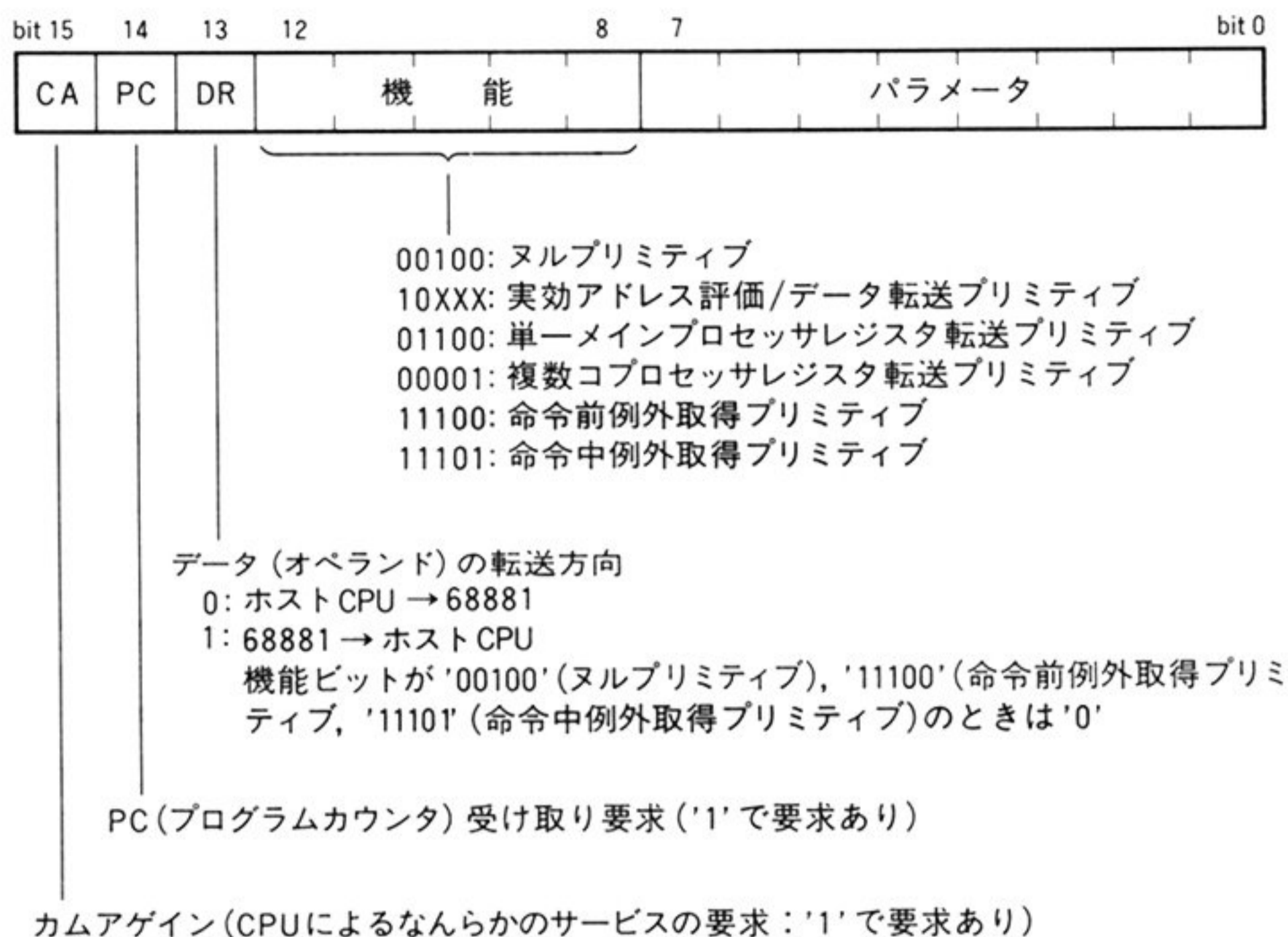
5 応答プリミティブ

応答プリミティブの一般的なフォーマットを 116 ページの図 9 に示します。

CA ビットは，68881 がなんらかのサービス要求を行っていることを示しています。

PC ビットは，ホスト CPU から PC（プログラムカウンタ）の値を受け渡すことを 68881 が要求しているときにセットされます。この要求は，X 68000 のように I/O デバイスとして使っているときには意味を持ちません。68881 側でも，そのような利用法を考慮し，この要求は無視されてもかまわないようになっています。

DR は，68881 とホスト CPU との間のデータ転送方向を示します。'0'のときにはホスト

●図……9　68881応答プリミティブのフォーマット

CPUから68881へ，'1'のときには68881からホストCPUへの転送であることを示します。

機能ビットはプリミティブの種別を示し，パラメータはそれぞれのプリミティブに付随した情報をホストCPUに渡すために使用されます。

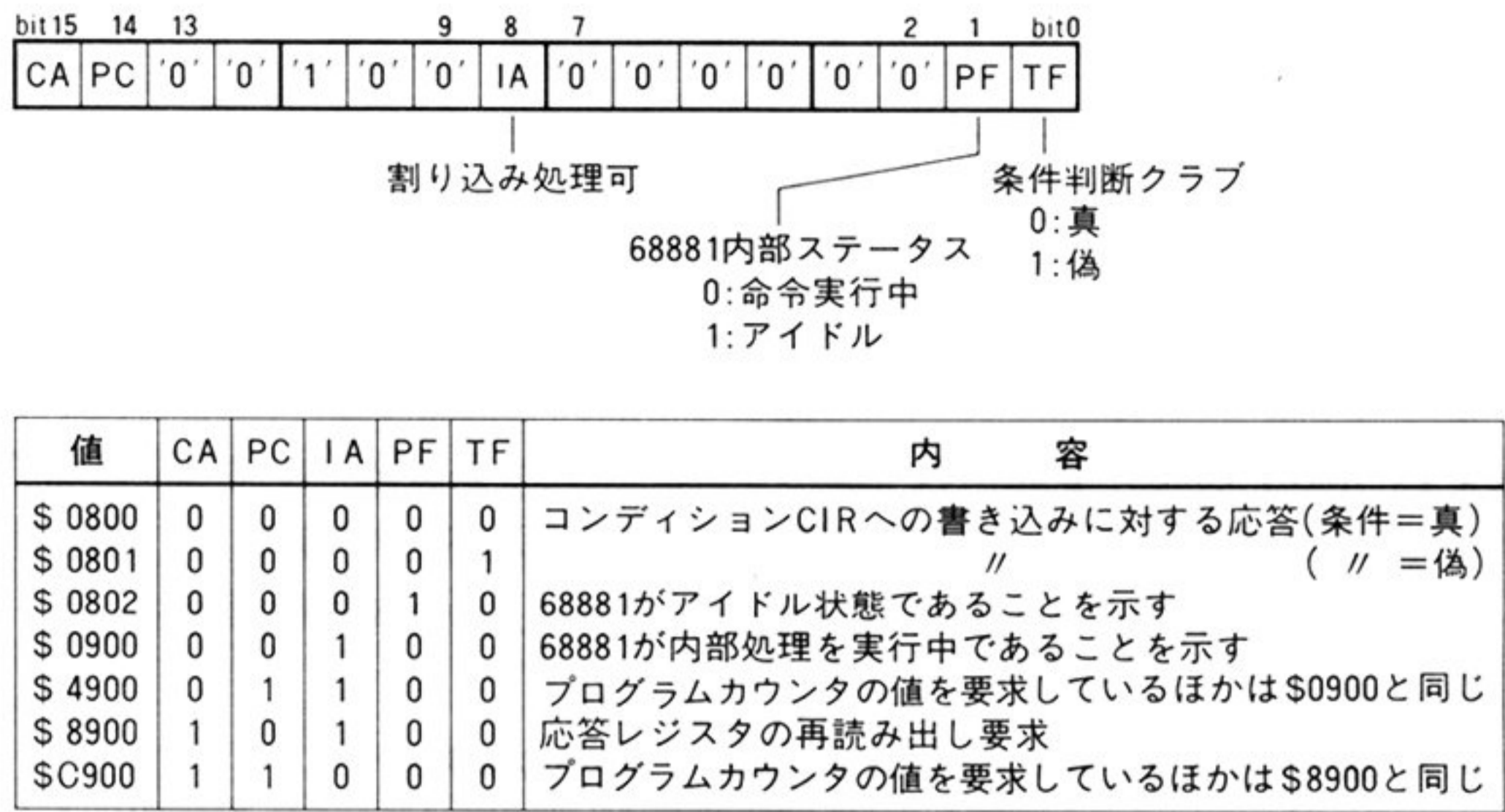
❺·1 ヌルプリミティブ

ヌルプリミティブの詳細を図10に示します。ヌルプリミティブは，68881が自身のステータスを知らせるとともに，ホストCPUとの同期をとるものです。ヌルプリミティブの各ビットの組み合わせとその内容の対応関係を図中に示しておきましたので，参考にしてください。68881は，これ以外の組み合わせの値を返すことはありません。

❺·2 実効アドレス評価/データ転送プリミティブ

実効アドレス評価/データ転送プリミティブは，68881がホストCPUに対して浮動小数点データやコントロールレジスタの値の転送要求を行うために使用されます。

●図……10 ヌルプリミティブの内容

値	CA	PC	IA	PF	TF	内　　容
$ 0800	0	0	0	0	0	コンディションCIRへの書き込みに対する応答(条件=真)
$ 0801	0	0	0	0	1	〃　　(〃 =偽)
$ 0802	0	0	0	1	0	68881がアイドル状態であることを示す
$ 0900	0	0	1	0	0	68881が内部処理を実行中であることを示す
$ 4900	0	1	1	0	0	プログラムカウンタの値を要求しているほかは$0900と同じ
$ 8900	1	0	1	0	0	応答レジスタの再読み出し要求
$C900	1	1	0	0	0	プログラムカウンタの値を要求しているほかは$8900と同じ

●図……11 実効アドレス評価/データ転送プリミティブの内容

bit15	14	13	12	11	10–8	7–bit0
'1'	PC	DR	'1'	'0'	有効<ea>	長さ

値	対応するオペレーション	PC	DR	有効<ea>	長さ	転送するデータ
$9501/$D501	浮動小数点演算命令	X	0	101	$01	バイト
$9502/$D502	FMOVE XX, FPm	X	0	101	$02	ワード
$9504/$D504		X	0	101	$04	ロングワード/単精度実数
$9508/$D508	(OPクラス:010)	X	0	110	$08	倍精度実数
$960C/$D60C		X	0	110	$0C	拡張精度実数/パック形式BCD
$B101	FMOVE FPm, XX	0	1	001	$01	バイト
$B102		0	1	001	$02	ワード
$B104		0	1	001	$04	ロングワード/単精度実数
$B208	(OPクラス:011)	0	1	010	$08	倍精度実数
$B20C		0	1	010	$0C	拡張精度実数/パック形式 BCD
$9704	FMOVE XX, FPcr	0	0	111	$04	転送するコントロールレジスタは4バイト
$9504	FMOVEM XX, FPcr-list	0	0	101	$04	〃 4 〃
$9608	(OPクラス:100)	0	0	110	$08	〃 8 〃
$960C		0	0	110	$0C	〃 12 〃
$B304	FMOVE FPcr, XX	0	1	011	$04	転送するコントロールレジスタは4バイト
$B104	FMOVEM FPcr-list, XX	0	1	001	$04	〃 4 〃
$B208	(OPクラス:101)	0	1	010	$08	〃 8 〃
$B20C		0	1	010	$0C	〃 12 〃

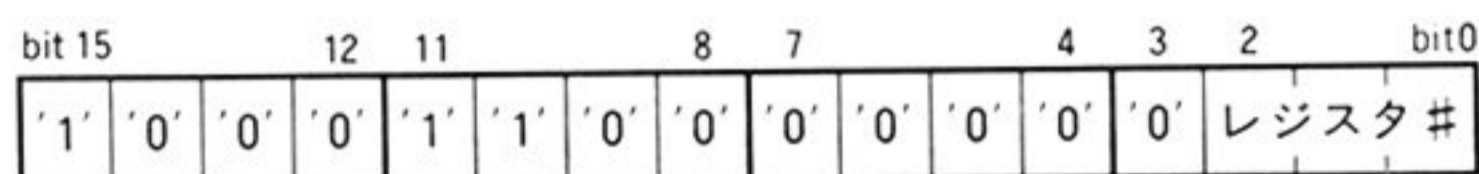
このプリミティブの詳細，プリミティブ値と対応する68881のオペレーション，転送するデータタイプの対応を図11に示します。このプリミティブは命令実行時に一度だけ返され，それ以降はヌルプリミティブに変化します。

❺・3 単一メインプロセッサレジスタ転送プリミティブ

プリミティブのフォーマットを図 12 に示します。

このプリミティブは，複数浮動小数点データレジスタ転送命令において，転送するレジスタリストをデータレジスタで指定するモードを選択した場合，68881 からメインプロセッサに要求されます。ホスト CPU が 68020 で，68881 が直結されている場合には，要求されたデータレジスタの内容が CPU によって自動的に転送されますが，X 68000 の場合には，次に書き込むデータが使用されるだけであり，レジスタ番号ビットはとくに意味を持ちません。

●図……12 単一メインプロセッサレジスタ転送プリミティブの内容

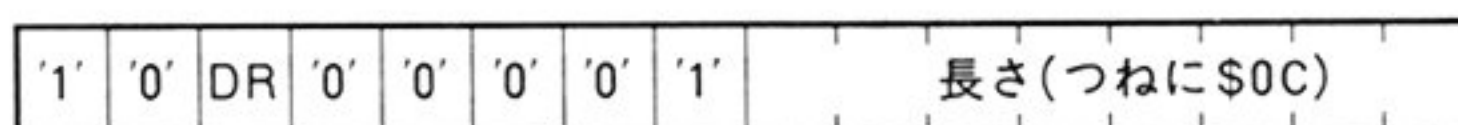
bit 15			12	11			8	7			4	3	2		bit0
'1'	'0'	'0'	'0'	'1'	'1'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	レジスタ#		

値	レジスタ#	転送するレジスタ
$8C00	000	D0
$8C01	001	D1
$8C02	010	D2
$8C03	011	D3
$8C04	100	D4
$8C05	101	D5
$8C06	110	D6
$8C07	111	D7

❺・4 複数コプロセッサレジスタ転送プリミティブ

複数コプロセッサレジスタ転送プリミティブのフォーマットを図 13 に示します。このプリミティブは，68881 が複数の浮動小数点データレジスタを外部との間で転送することを要求するために使用します。

長さフィールドは転送される各レジスタのバイト数を示しますが，68881 の場合にはつねに $0 C になります。

●図……13　複数コプロセッサレジスタ転送プリミティブの内容

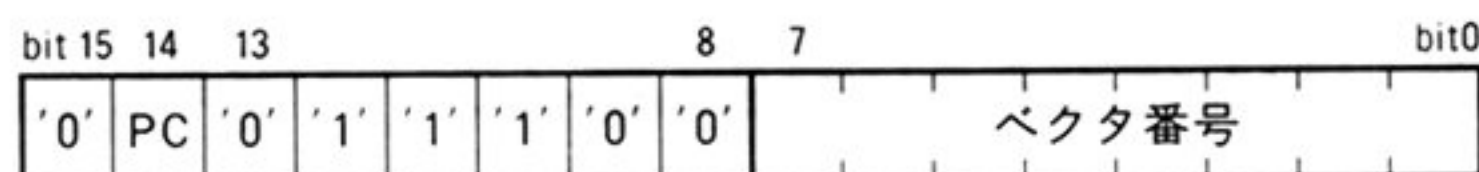
'1'	'0'	DR	'0'	'0'	'0'	'0'	'1'	長さ(つねに$0C)

値	DR	転送方向
$810C	0	メモリから68881への転送
$A10C	1	68881からメモリ　〃

❺·5 命令前例外取得プリミティブ/命令中例外取得プリミティブ

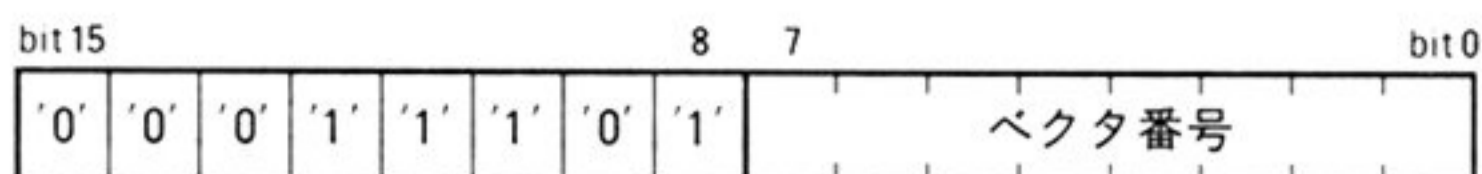
命令前例外取得プリミティブは，68881がなんらかの異常を検出し，ホストCPUに対して現在のオペレーションをアボートし，例外処理の開始を要求するために使用します。命令中例外取得プリミティブは，浮動小数点データレジスタから外部への転送命令の実行中に例外が発生したときに通知されるプリミティブです。

それぞれのプリミティブのフォーマットを図14と図15に示します。下位8ビットのベクタビットは，発生した例外状態を示すのに使用されます。68881が発生するベクタ番号と，その内容の対応も図中に示しておきましたので参考にしてください。

●図……14　命令前例外取得プリミティブの内容

bit 15	14	13					8	7 … bit0
'0'	PC	'0'	'1'	'1'	'1'	'0'	'0'	ベクタ番号

値	PC	ベクタ番号	内　容
$5C0B	1	$0B	Fラインエミュレータ
$5C30	1	$30	アンオーダ条件での分岐 /セット
$1C31	0	$31	不正確な結果
$1C32	0	$32	ゼロによる浮動小数点の除算
$1C33	0	$33	アンダフロー
$1C34	0	$34	オペランドエラー
$1C35	0	$35	オーバフロー
$1C36	0	$36	シグナリングNAN

●図……15　命令中例外取得プリミティブの内容

bit15							8	7 ～ bit0
'0'	'0'	'0'	'1'	'1'	'1'	'0'	'1'	ベクタ番号

値	ベクタ番号	内　容
$1D0D	$0D	コプロセッサプロトコル違反
$1D31	$31	不正確な結果
$1D32	$32	ゼロによる浮動小数点除算
$1D33	$33	アンダフロー
$1D34	$34	オペランドエラー
$1D35	$35	オーバフロー
$1D36	$36	シグナリングNAN

●6　68881とホストCPUのコミュニケーション

応答プリミティブは種類が多く，68881とホストCPUとのコミュニケーションは厄介なように思えますが，実際には命令ごとに応答されるプリミティブの種類はほぼ決まっているため，考えなくてはならない応答の種類はそれほど多くありません。

❻·1　68881内レジスタ間演算/データ転送命令

68881内部の浮動小数点データレジスタどうしでの演算やデータ転送（FADD.X FP0,FP1など）の手順を図16に示します。

まず，CPUがこれらの命令をコマンドCIRに書き込みます。図中，アクセスするレジスタの欄は，そのオペレーションでホストCPUがアクセスするレジスタを示しており，応答CIRの欄はその時点での応答CIRの値を示しています。

68881は，ホストCPUに対して，応答CIRに$0900か$4900（ヌルプリミティブ：内部処理実行中)をセットします。このプリミティブのPCビットがセットされているとき，68881はホストCPUに現在のPC（プログラムカウンタ）の値の書き込みを要求しているわけですが，これはX68000のような使い方の場合にはとくに意味を持ちませんので，無視してしまってかまいません（書き込んでもエラーにはなりませんが)。

●図……16　ホスト CPU と 68881 のコミュニケーション（その１）

68881内レジスタ間演算/データ転送命令（OPクラス：000）

ホストCPUの動作	68881の動作	アクセスするレジスタ	応答CIRの値
	（アイドル）		$0802
コマンド書き込み →		コマンドCIR	
応答読み出し ←		応答CIR	$0900/$4900
PC書き込み(省略可) →	演算	命令アドレスCIR	$0900
	丸め		
	（アイドル）		$0802

CPU が応答 CIR を読み出すと，68881 は動作を開始し，演算，丸め処理を実行し，データを指定された浮動小数点データレジスタに格納します。

命令の実行が終了すると，68881 は応答 CIR を$0802（ヌルプリミティブ：アイドル状態）として，ホスト CPU から次の要求がくるのを待ちます。

❻·2 レジスタと外部データの間の演算/外部からレジスタへのデータ転送命令

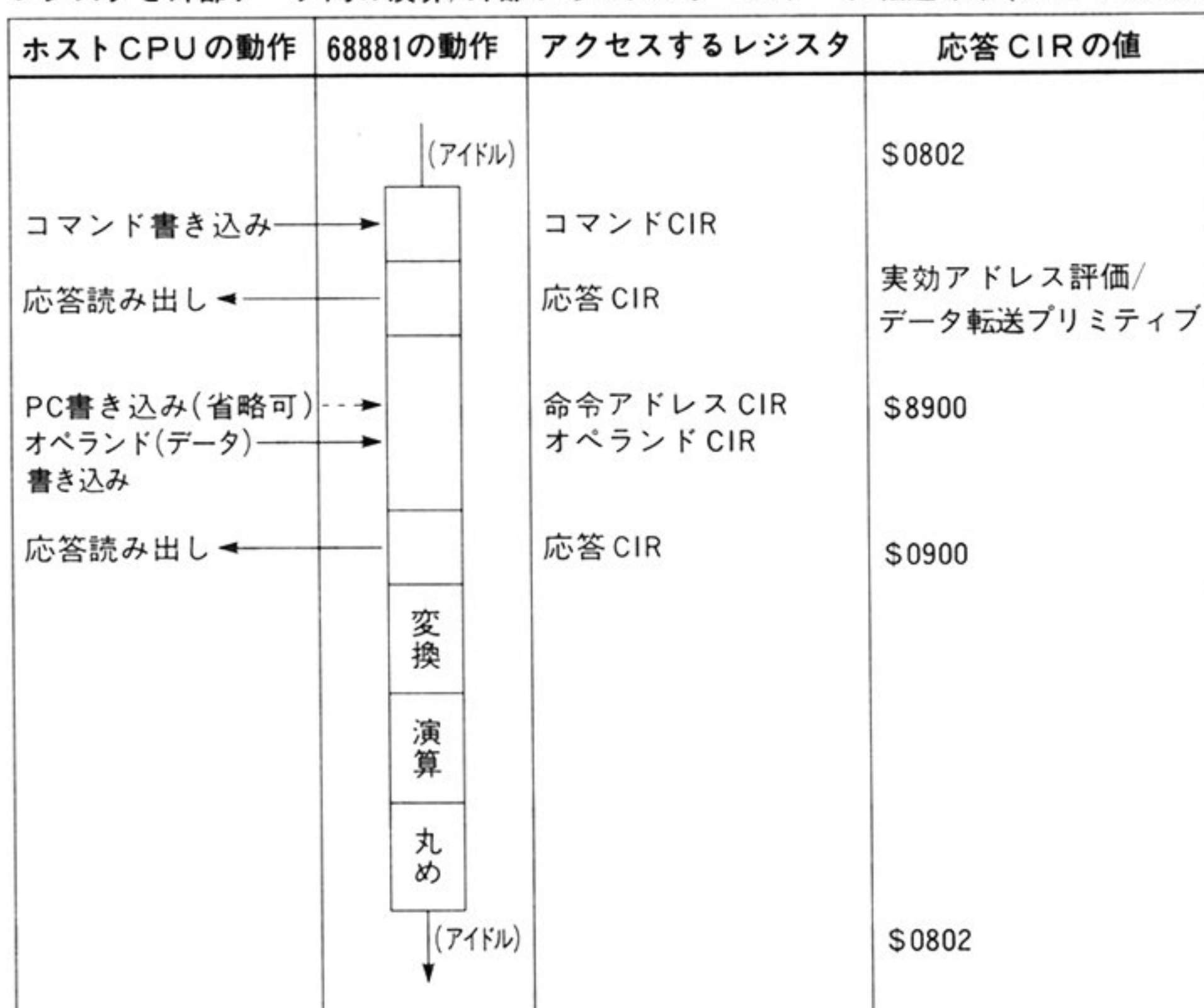
浮動小数点データレジスタと外部から書き込まれるデータとの間の演算（FADD.S #32,FP 0 など）や，外部から浮動小数点データレジスタへのデータ転送（FMOVE.S #11,FP 2 など）の手順を，122 ページの図 17 に示します。

まず，ホスト CPU はコマンド CIR に演算命令やデータ転送命令を書き込みます。68881 は，この命令に対し，実行アドレス評価/データ転送プリミティブを応答 CIR にセットし，ホスト CPU に対してデータ転送を要求します。

次にホスト CPU は，68881 が要求しているデータをオペランド CIR 経由で 68881 に転送します。転送が終了すると，68881 は応答 CIR の値を$0900（ヌルプリミティブ：内部処理実行中）とし，データの型変換，演算，丸め処理などを行い，結果を命令で指定された浮動小数点データレジスタに転送します。

●図……17　ホストCPUと68881のコミュニケーション(その2)

レジスタと外部データ間の演算/外部からレジスタへのデータ転送命令(OPクラス:010)

ホストCPUの動作	68881の動作	アクセスするレジスタ	応答CIRの値
	(アイドル)		$0802
コマンド書き込み→		コマンドCIR	
応答読み出し←		応答CIR	実効アドレス評価/データ転送プリミティブ
PC書き込み(省略可)→ オペランド(データ)書き込み→		命令アドレスCIR オペランドCIR	$8900
応答読み出し←		応答CIR	$0900
	変換		
	演算		
	丸め		
	(アイドル)		$0802

⑥·3 レジスタから外部へのデータ転送

68881内部の浮動小数点データレジスタの読み出しの手順を図18に示します。

データフォーマットとしてパック形式10進データを指定した場合，データ形式(小数点以下の桁数など)の指定が必要になります。このデータ形式をKファクターと呼びます。Kファクターを命令中に含めてしまうのが静的（スタティック）Kファクター，データとして別途与えるのが動的（ダイナミック）Kファクターです。

レジスタから外部への転送手順は，ダイナミックKファクターが使用される場合と，それ以外の場合に区別されます。

まず，通常の転送では，コマンドCIRにコマンドを書き込むと，応答CIRとして$8900か$C900(ヌルプリミティブ：応答レジスタの再読み出し要求)を返し，68881内部のデータからコマンドで指定されたフォーマットへの変換動作を開始します。

変換が終了すると，応答CIRは実行アドレス評価/データ転送プリミティブに変わります。ホストCPUは，68881の，この応答を待って，オペランドCIRからデータを読み出します。

●図……18　ホスト CPU と 68881 のコミュニケーション(その 3)

レジスタから外部へのデータ転送(OPクラス:011)ダイナミックKファクターなしの場合

ホストCPUの動作	68881の動作	アクセスするレジスタ	応答CIRの値
	(アイドル)		$0802
コマンド書き込み →		コマンドCIR	
応答読み出し ←	変換	応答CIR	$8900/$C900
PCの書き込み(省略可) --→	変換	命令アドレスCIR	
応答読み出し ←		応答CIR	実効アドレス評価/データ転送プリミティブ
レジスタ読み出し ←		オペランドCIR	
応答読み出し ←		応答CIR	$0802
	(アイドル)		

レジスタから外部へのデータ転送(OPクラス:011)ダイナミックKファクターありの場合

ホストCPUの動作	68881の動作	アクセスするレジスタ	応答CIRの値
	(アイドル)		$0802
コマンド書き込み →		コマンドCIR	
応答読み出し ←		応答CIR	単一メインプロセッサレジスタ転送プリミティブ
PC書き込み(省略可) --→		命令アドレスCIR	
レジスタ転送 →		オペランドCIR	$8900
応答読み出し ←	変換	応答CIR	$8900
レジスタ読み出し ←		応答CIR	実効アドレス評価/データ転送プリミティブ
オペランド(データ)転送 ←		オペランドCIR	
応答読み出し ←		応答CIR	$0802
	(アイドル)		

読み出しが終了したら，ホストCPUは応答CIRを再度読み出します。このときの応答CIRは$0802（ヌルプリミティブ：アイドル状態）となっています。

ダイナミックKファクターが指定された場合，最初の応答プリミティブでは，単一メインプロセッサレジスタ転送プリミティブが返されます。ホストCPUは，この応答を見てオペランドCIR経由でKファクター値を68881に転送します。

68881は，Kファクター値を受け取ると，応答CIRを$8900（ヌルプリミティブ：内部処理実行中）とし，内部データの変換動作を開始します。

変換が終了した後の動作は先ほどの通常の転送と同じで，実行アドレス評価/データ転送プリミティブを待った後，データの転送を実行し，最後に応答CIRでヌルプリミティブを受け取って終了します。

❻·4 コントロールレジスタの転送命令

FPCR，FPSR，FPIARの1つ，あるいは複数を転送するのがこの転送命令動作です。転送手順を図19に示します。基本的には浮動小数点データ転送と大差ありませんが，データ変換動作が不要な分，かんたんになっています。

コマンドを書き込んだ後，応答CIRとして実行アドレス評価/データ転送プリミティブが返されます。ホストCPUは，これを受け取った後，データの転送を行います。転送するコントロールレジスタとして複数のレジスタが指定されている場合には，このデータ転送が何度か繰り返されることになります。

転送が終了したら，応答CIRを読み出します。このときの値としては$0802（ヌルプリミティブ：アイドル状態）が返ってきます。

❻·5 複数浮動小数点データレジスタの転送

複数浮動小数点データレジスタ転送動作を126ページの図20に示します。

複数浮動小数点データレジスタ転送は，転送するレジスタの指定を命令中に含める場合（スタティックレジスタリスト）と，パラメータとして与える場合（ダイナミックレジスタリスト）の2通りがあります。

スタティックレジスタリストの場合，最初の応答CIRとして，複数コプロセッサレジスタ転送CIRが返されます。ホストCPUは，この後，レジスタ選択CIRを読み出し，'1'になっているビットの数を数えることで転送するレジスタの数を把握します。これをもとに，ホスト

●図……19　ホストCPUと68881のコミュニケーション(その4)

コントロールレジスタ転送(読み出し)命令(OPクラス:100)

ホストCPUの動作	68881の動作	アクセスするレジスタ	応答CIRの値
	(アイドル)		$0802
コマンド書き込み→		コマンドCIR	
応答読み出し←		応答CIR	実効アドレス評価/データ転送プリミティブ $8900
レジスタの転送←		オペランドCIR	
応答読み出し←		応答CIR	
	(アイドル)		$0802

コントロールレジスタ転送(書き込み)命令(OPクラス:101)

ホストCPUの動作	68881の動作	アクセスするレジスタ	応答CIRの値
	(アイドル)		$0802
コマンド書き込み→		コマンドCIR	実効アドレス評価/データ転送プリミティブ
応答読み出し←		応答CIR	$8900
オペランド(データ)転送→		オペランドCIR	
応答読み出し←		応答CIR	
	(アイドル)		$0802

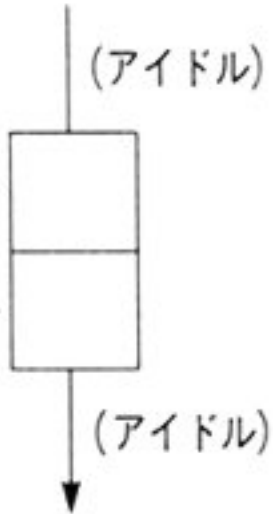
CPUは68881とオペランドCIR経由でデータの転送を行い，最後に応答CIRを読み出して終了します。

ダイナミックレジスタリストを使用した場合，最初の応答としては単一メインプロセッサレジスタ転送プリミティブが返され，ホストCPUからレジスタリストの転送を要求します。ホストCPUは68881にレジスタリストを渡します。これ以降の動作は，スタティックレジスタリストの場合と同一です。

●図……20　ホストCPUと68881のコミュニケーション(その5)

複数浮動小数点データレジスタの転送(ダイナミックレジスタリスト)

ホストCPUの動作	68881の動作	アクセスするレジスタ	応答CIRの値
	(アイドル)		$0802
コマンド書き込み→		コマンドCIR	
応答読み出し←		応答CIR	単一メインプロセッサレジスタ転送プリミティブ
レジスタリスト書き込み→		オペランドCIR	$8900
応答読み出し←		応答CIR	複数コプロセッサレジスタ転送プリミティブ
レジスタマスク読み出し←		レジスタ選択CIR	$8900
レジスタ転送←→		オペランドCIR	
応答読み出し←		応答CIR	$0802
	(アイドル)		

複数浮動小数点データレジスタの転送(スタティックレジスタリスト)

ホストCPUの動作	68881の動作	アクセスするレジスタ	応答CIRの値
	(アイドル)		$0802
コマンド書き込み→		コマンドCIR	
応答読み出し←		応答CIR	複数コプロセッサレジスタ転送CIR
レジスタマスク読み出し←		レジスタ選択CIR	$8900
レジスタ転送←→		オペランドCIR	
応答読み出し←			$0802
	(アイドル)		

❻·6 条件付き命令処理動作

条件付き命令というのは，条件分岐（Bcc）などの命令の総称です。68881が68020と直結

●図……21　ホストCPUと68881のコミュニケーション(その6)

条件付き命令処理動作

ホストCPUの動作	68881の動作	アクセスするレジスタ	応答CIRの値
	(アイドル)		
コンディション書き込み→		コンディションCIR	
応答読み出し←		応答CIR	$0800/$0801
	(アイドル)		

されている場合には，68020は68881から返されたステータスをもとに分岐などを行いますが，X68000のような使い方では，このような処理がCPUによって行われることはなく，たんなるステータスチェック命令として使うよりありません。

この命令のコミュニケーション手順を図21に示します。最初のアクセスでコマンドCIRではなく，コンディションCIRを使うことに注意してください。

応答CIRとして返ってくるのはヌルプリミティブです。指定した条件が成立した場合には$0800，成立しなかった場合には$0801が返されます。

❻·7　FSAVE/FRESTORE命令処理動作

68881の内部ステータスのセーブ/リストアを行う命令です。この命令の処理手順を128ページの図22に示します。

FSAVE命令の場合，セーブCIRの読み出し動作から転送動作が開始されます。このとき返ってくる値としては，次の4種類があります。

$0018	：NULLステート	(転送するデータはなし)
$0118	：カムアゲイン	
$XX18	：アイドルステート	(転送するデータは24($18)バイト)
$XXB4	：ビジーステート	(転送するデータは180($B4)バイト)

XXは68881のバージョンを示します。カムアゲインが返ってきた場合，ホストCPUは再度セーブCIRの読み出しを行い，カムアゲイン以外のステータスが返ってくるのを待ちます。

カムアゲイン以外のステータスが返ってきたら，ホストCPUは，各フォーマットごとに必

●図……22　ホスト CPU と 68881 のコミュニケーション(その 7)

FSAVE 命令処理動作

ホストCPUの動作	68881の動作	アクセスするレジスタ	応答CIRの値
セーブCIRの読み出し		セーブCIR	$0802
$0018:NULLステート $XX18:IDLE 〃 $XXB4:BUSY 〃 $0118: カムアゲイン			
オペランド(データ)転送		オペランドCIR	
	(アイドル)		

FRESTORE処理動作

要な量のデータ読み出しを行い，68881 の内部ステータスを保存します。

FRESTORE 命令は，ちょうどこれとは逆の動作です。リストア動作は，リストア CIR にセーブ CIR 読み出し時に受け取ったステート情報を書き込むことからスタートします。この後，リストア CIR を再度読み出し，NULL，IDLE，BUSY のいずれかのステートが入っているのを確認して，セーブしておいた 68881 の内部ステートの書き込みを行います。

❻・8 例外処理動作

命令の実行時，68881がなんらかの異常を見つけたときの動作が例外処理動作です。例外動作には命令前例外処理動作，BSUN例外動作，Fラインエミュレータ例外(68881が実行できない命令を受け取ったときの)動作，FSAVEフォーマット例外(FSAVE命令動作中にFSAVE命令を実行しようとしたときの)動作，FRESTOREフォーマット例外(リストアCIRに書

●図……23　ホストCPUと68881のコミュニケーション(その8)

命令前例外処理動作

ホストCPUの動作	68881の動作	アクセスするレジスタ	応答CIRの動作
	(アイドル)		$0802
コマンド書き込み→		コマンドCIR	
応答読み出し←		応答CIR	命令前例外取得プリミティブ
例外アクノリッジの書き込み→		コントロールCIR	
	(アイドル)		$0802

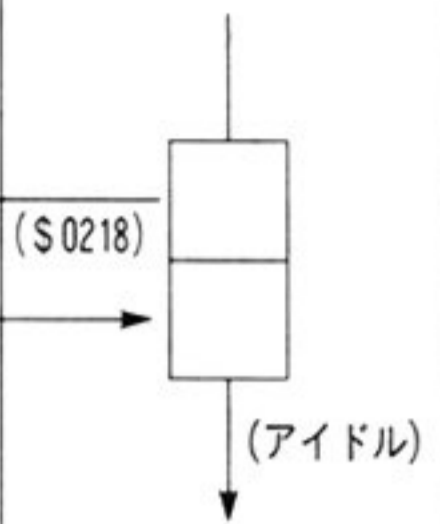
命令中例外処理動作

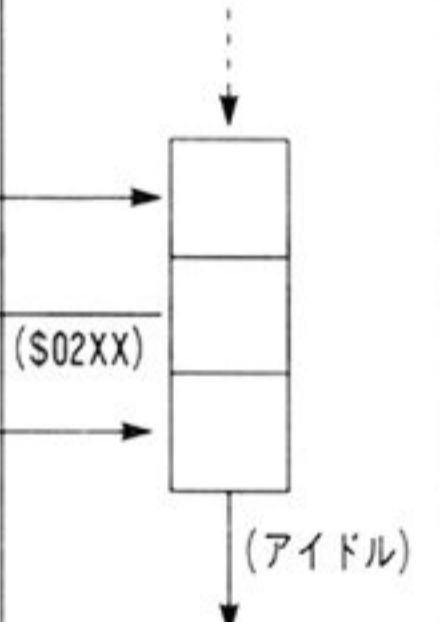
●図……24　ホストCPUと68881のコミュニケーション(その9)

BSUN例外動作

ホストCPUの動作	68881の動作	アクセスするレジスタ	応答CIRの値
	(アイドル)		$0802
コンディション書き込み		コンディションCIR	
応答の読み出し		応答CIR	命令前例外取得
PCの書き込み(省略可)		命令アドレスCIR R	
例外アクノリッジ書き込み		コントロール CIR R	
	(アイドル)		$0802

●図……25　ホストCPUと68881のコミュニケーション(その10)

Fラインエミュレータ例外動作

ホストCPUの動作	68881の動作	アクセスするレジスタ	応答CIRの値
	(アイドル)		$0802
コマンド書き込み		コマンドCIR	
応答読み出し		応答CIR	命令前例外取得プリミティブ
PC書き込み(省略可)		命令アドレスCIR	
例外アクノリッジの書き込み		コントロールCIR	
	(アイドル)		$0802

き込んだデータフォーマットがおかしいときの）動作などがあります。それぞれの処理手順を図23，図24，図25，図26に示します。

いずれの場合も，ホストCPUは，68881からの例外通知を受け取った後，コントロールCIRへの書き込みを行い，68881をアイドル状態に復帰させます。

●図……26　ホストCPUと68881のコミュニケーション(その11)

FSAVEフォーマット例外処理動作

ホストCPUの動作	68881の動作	アクセスするレジスタ	応答CIRの値
			(前のFSAVE/FRESTORE命令処理中)
セーブCIR読み出し ←	($0218)	セーブCIR	
アボート書き込み →		コントロールCIR	
	(アイドル)		$0802

FRESTOREフォーマット例外処理動作

ホストCPUの動作	68881の動作	アクセスするレジスタ	応答CIRの値
リストアCIR書き込み →		リストアCIR	
リストアCIR読み出し ←	($02XX)	リストアCIR	
アボート書き込み →		コントロールCIR	
	(アイドル)		$0802

7 68881の命令フォーマット

68881の命令の大部分は，コマンドCIRを利用して与えるものです。ここでは，このようにして利用できる命令の説明を行うことにします。

7·1 一般的な命令(OPクラス000/010)

68881の命令フォーマットは，その上位3ビットで大きく分類でき，この3ビットをOPクラスと呼んでいます。68881で通常使用する演算命令や，外部から68881へのデータ転送は，

すべてOPクラス000と010に分類されます。OPクラス000はレジスタどうし，010はレジスタと外部データの間の演算や外部からレジスタへの転送命令を示します。

これらの命令のフォーマットを図27に示します。

R/Mビットが，ソースデータがレジスタか，外部から与えられるデータであるかを示すもので，'0'のときレジスタ，'1'のときに外部データとなります。ソースフィールドは，このR/Mビットによって意味が変わります。R/Mが'0'のときには，ソースフィールドはレジスタ番号を，R/Mが'1'のときには与えるデータの型を指定するために使用します。R/Mが'1'でソースフィールドが'111'のときはFMOVECR命令（次に説明します）になるため，機能フィールドの意味が変わります。

ディスティネーションレジスタ#フィールドは，演算の対象や演算結果の格納先として使われる浮動小数点データレジスタ番号を示します。

機能フィールドは演算命令の指定に使用します。機能フィールドの値と行われる演算の関係を図の中に示しておきましたので，参考にしてください。

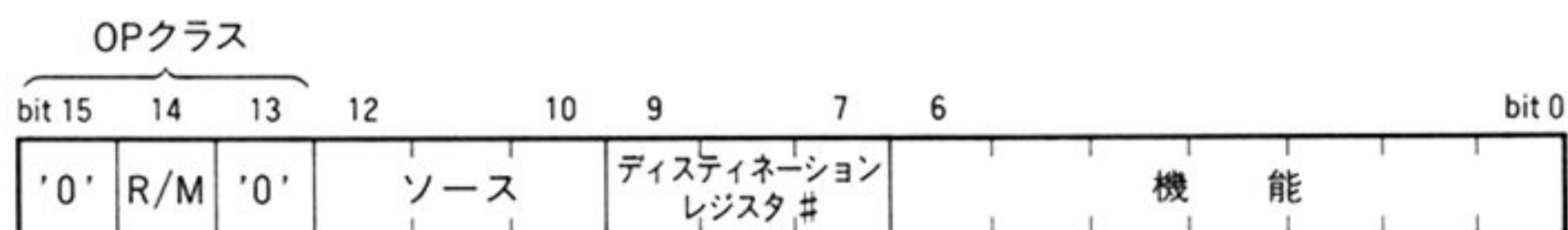
❼・2 FMOVECR（Move from Constant Rom）命令

68881は，円周率や自然対数の底（2.71828……）など，数値演算のときによく使用される定数値をあらかじめチップ内部のROMに持っています。これを読み出し，浮動小数点データレジスタに転送するのがFMOVECR命令です。FMOVECR命令の命令フォーマットを134ページの図28に示します。上位の9ビットは，先ほどの一般的な命令のR/Mフィールドを'1'，ソースフィールドを'111'としたビットパターンにあたります。

下位7ビットは，68881内部の定数ROMのオフセット（定数の番号と呼んだほうが適切かもしれません）を指定します。オフセット値と格納されている定数値の対応を図中に整理しておきましたので参考にしてください。表に載っていないオフセット値のところにもなにがしかのデータが入っていますが，これは68881のマイクロコードが使用するためのもので，ユーザには開放されていません（将来，内容が変更されないという保証もありません）。

❼・3 浮動小数点レジスタから外部への転送

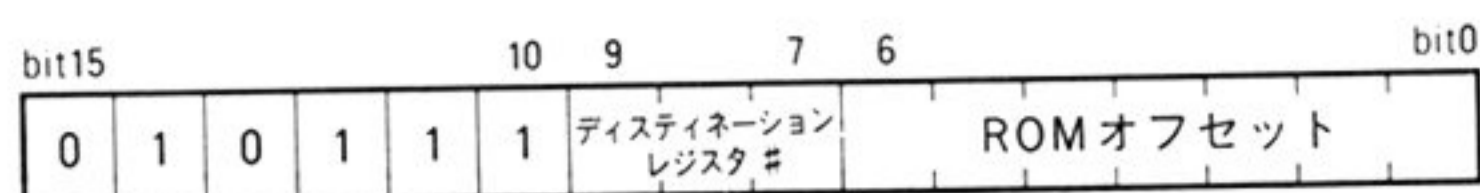
浮動小数点レジスタから外部への転送命令（FMOVE FP0,XXなど）の命令フォーマットを135ページの図29に示します。

●図……27　68881の命令フォーマット（OPクラス：000/010）

R/M	ソース	ソースオペランド	略号
'0'	000	FP0	
	001	FP1	
	010	FP2	
	011	FP3	
	100	FP4	
	101	FP5	
	110	FP6	
	111	FP7	
'1'	000	ロングワード整数	L
	001	単精度実数	S
	010	拡張精度実数	X
	011	パック型式10進実数	P
	100	ワード整数	W
	101	倍精度実数	D
	110	バイト整数	B
	111	<FMOVECR 命令>	

機能	命　令	
$00	FMOVE to FP_n	データ転送
$01	FINT	整数部分の取り出し
$02	FSINH	SINH（双曲SIN）
$03	FINTRZ	整数部分の取り出し（0に丸める）
$04	FSQRT	平方根
$06	FLOGNP1	Log (x+1)
$08	FETOXM1	e^x-1
$09	FTANH	TANH（双曲TAN）
$0A	FATAN	TAN^{-1}（アークTAN）
$0C	FASIN	SIN^{-1}（アークSIN）
$0D	FATANH	$TANH^{-1}$（双曲アークTAN）
$0E	FSIN	SIN
$0F	FTAN	TAN
$10	FETOX	e^x
$11	FTWOTOX	2^x
$12	FTENTOX	10^x
$14	FLOGN	Log
$15	FLOG10	Log_{10}
$16	FLOG2	Log_2

機能	命　令	
$18	FABS	絶対値
$19	FCOSH	COSH（双曲コサイン）
$1A	FNEG	－X（補数）
$1C	FACOS	COS^{-1}（アークCOS）
$1D	FCOS	COS
$1E	FGETEXP	指数部の取り出し
$1F	FGETMAN	仮数部の取り出し
$20	FDIV	除算
$21	FMOD	モジュロ剰余
$22	FADD	加算
$23	FMUL	乗算
$24	FSGLDIV	単精度除算
$25	FREM	剰余（IEEE形式）
$26	FSCALE	FP_n × INT (2^x)
$27	FSGLMUL	単精度乗算
$28	FSUB	減算
$30 ～ $37	FSINCOS	SINとCOSを同時に求める（下位3bitでCOSを入れるレジスタ選択）
$38	FCMP	比較
$3A	FTST	オペランドのテスト
$40 ～ $7F	（未使用）	

●図……28　FMOVECR（定数データの転送）

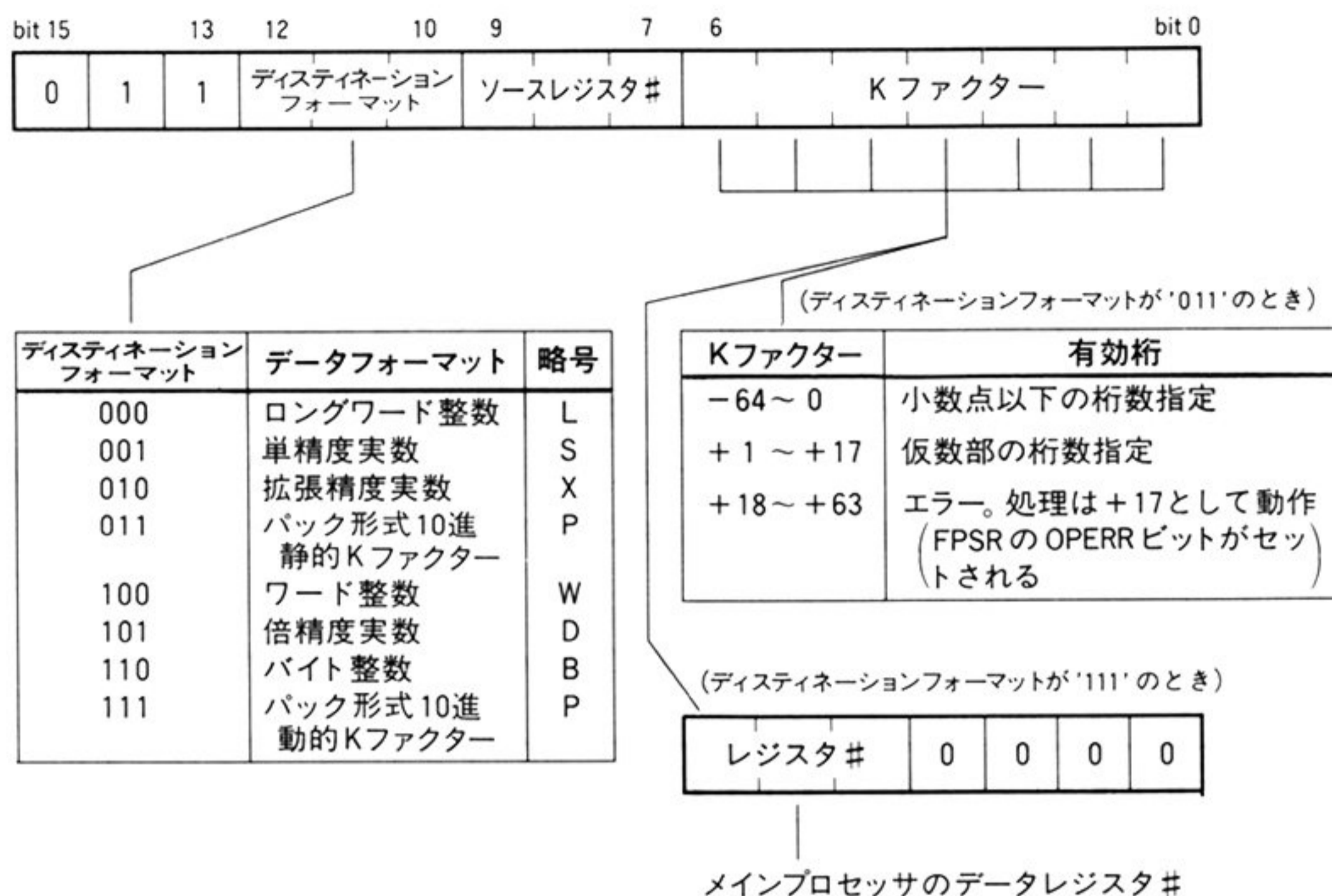
bit15					10	9　　7	6　　　　　　　　bit0
0	1	0	1	1	1	ディスティネーションレジスタ#	ROMオフセット

ROMオフセット値	格納されている値
$00	π
$0B	$Log_{10}2$
$0C	e
$0D	$Log_2 e$
$0E	$Log_{10} e$
$0F	0.0
$30	$Log_e 2$
$31	$Log_e 10$
$32	10^0
$33	10^1
$34	10^2
$35	10^4
$36	10^8
$37	10^{16}
$38	10^{32}
$39	10^{64}
$3A	10^{128}
$3B	10^{256}
$3C	10^{512}
$3D	10^{1024}
$3E	10^{2048}
$3F	10^{4096}

ビット7～9で転送元の浮動小数点データレジスタの番号を，ビット10～12の3ビットで外部に出力されるデータフォーマット(型)を指定します。68881は，浮動小数点データレジスタのデータ（通常は拡張精度です）を指定された型に変換して出力してきます。

下位7ビットは，データフォーマットとしてパック形式10進（BCD）を指定したときに仮数部や小数部の桁数指定を行うために使用するものです。前に書いたように，この指定をKファクターと呼び，Kファクターの指定を命令中に含めるのをスタティックKファクター，別途データとして与えるのをダイナミックKファクターと呼びます。

Kファクターフィールドは，データフォーマットがパック形式10進以外のときにはすべて'0'としてください。

データフォーマットが'011'，すなわち，スタティックKファクターの場合，Kファクターフィールドは桁数の指定を行うデータが入ります。このデータが0，あるいは負である場合には，ソースデータの小数部分の桁数を指定し，正の数の場合には仮数部分の桁数を指定します。たとえば，ソースデータが3141.59265のとき，Kファクターが−3だと小数点以下3桁，すなわち3141.593（丸めが行われるため，最下位桁が3になります）となり，これが正規化されて3.141593 E+3が返ってきます。同様に，Kファクターが0だと，3.142 E+3となります。

●図……29　FMOVE　FPn, XX（浮動小数点データレジスタから外部への転送）

Kファクターが+3のときには，仮数部の桁数が3桁ということですから，3.14 E+3，+5なら3.1415 E+3というぐあいになります。

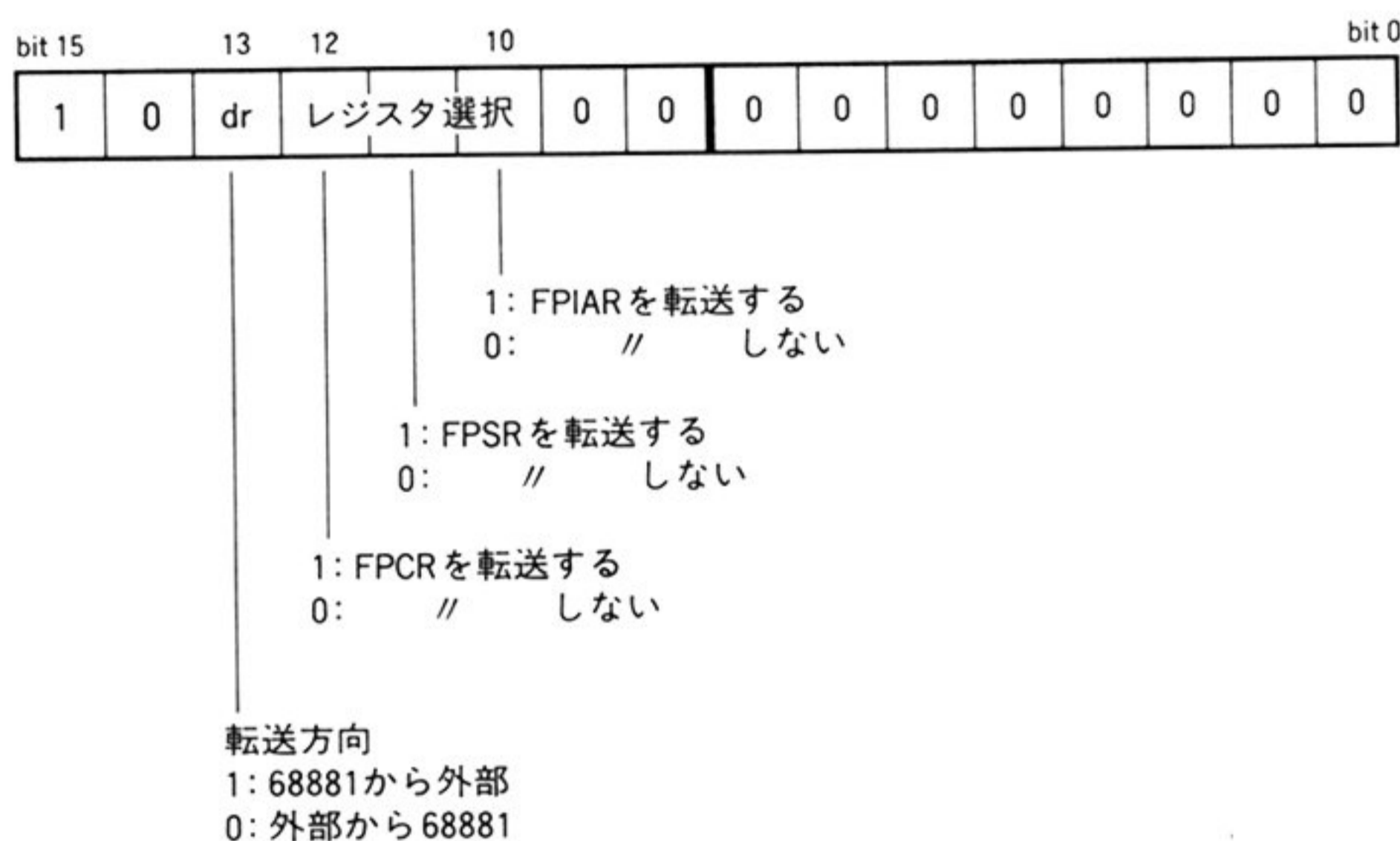
データフォーマットが'111'，すなわち，ダイナミックKファクターの場合には，Kファクターフィールドはkファクターのデータが入ったメインプロセッサのレジスタ（D 0〜D 7）番号を指定します。これは68881が68020と直結されているときに有効なもので，X 68000の場合にはとくに意味はありません（コミュニケーション手順のところも参照してください）。

ダイナミックKファクターを使用した場合，68881にKファクターデータを別途引き渡さなくてはなりません。この手順については，先に説明したコミュニケーション手順のところを参照してください。

❼·4 コントロールレジスタの転送

68881が持っているFPCR, FPSR, FPIARの各レジスタの転送を行うのがこの命令です。命令のフォーマットを図30に示します。

ビット12, 11, 10がそれぞれFPCR, FPSR, FPIARに対応し，'1'になっているレジスタが転送対象となります。ニーモニック上は単一コントロールレジスタの転送を行うFMOVE FPcrと，複数のコントロールレジスタの転送を行うFMOVEM FPcrに分かれますが，命令フォーマットはどちらも同じで，たんにビット12, 11, 10のうち，どれか1つしか'1'になっていないか，複数が'1'になっているかというだけのことです。

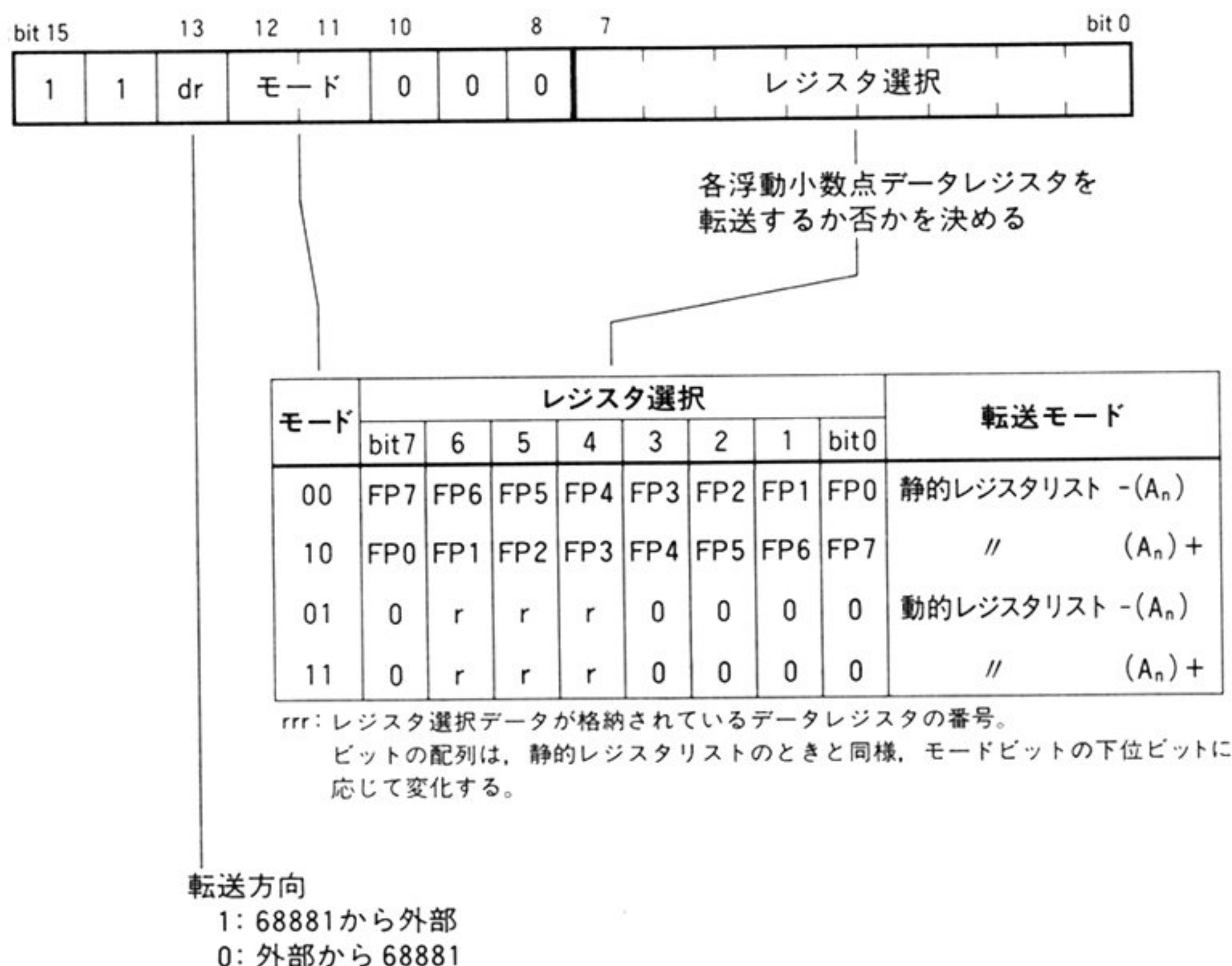
●図……30　コントロールレジスタの転送（FMOVE FPcr/FMOVEM FPcr）

❼·5 複数浮動小数点データレジスタの転送

68000のMOVEM命令に相当するのが，複数の浮動小数点データレジスタの転送命令(FMOVEM)です。命令フォーマットを図31に示します。

8本の浮動小数点データレジスタのどれを転送し，どれを転送しないかは，レジスタ選択フィールドで指定する方法（静的レジスタリスト）と，別途データとして与える方法（動的レジスタリスト）のいずれかで選択できます。さらに，レジスタリストの各ビットと各レジスタの対応が2通りずつあります。複数のデータレジスタを転送する場合，68000ではポストインクリ

●図……31　複数浮動小数点データレジスタの転送（MOVEM FPn）

モード	レジスタ選択								転送モード
	bit7	6	5	4	3	2	1	bit0	
00	FP7	FP6	FP5	FP4	FP3	FP2	FP1	FP0	静的レジスタリスト $-(A_n)$
10	FP0	FP1	FP2	FP3	FP4	FP5	FP6	FP7	〃 $(A_n)+$
01	0	r	r	r	0	0	0	0	動的レジスタリスト $-(A_n)$
11	0	r	r	r	0	0	0	0	〃 $(A_n)+$

rrr：レジスタ選択データが格納されているデータレジスタの番号。
ビットの配列は，静的レジスタリストのときと同様，モードビットの下位ビットに応じて変化する。

メント（(A 0+)など）やプリデクリメント（(−A 0)など）のアドレッシングモードを使用するのが一般的ですが，この両者ではデータ転送を行う順序が逆になります（プリデクリメントでFP 0，FP 1，FP 2の順序で待避したものをポストインクリメントで取り出すときはFP 2，FP 1，FP 0の順で読み出される)。68881は，どちらの順序でのデータ入出力も可能なようにしているわけです。

これらの組み合わせで得られる計4通りの転送モードのいずれを使用するかを選択するのがモードフィールド，データの転送方向を決めるのがdrビットです。

❼·6 条件付き命令のフォーマット

条件付き命令とは条件分岐命令などの総称ですが，X 68000の場合には68881をI/Oデバイスとして接続していますので，この命令はたんに条件を与えて前回までの演算結果がそれと一致するか否かをチェックするだけの命令になります。

この命令のフォーマットを138ページの図32に示します。

下位6ビット(コンディションフィールド)で条件を与えると，68881はそれと演算の結果得

られているフラグを比較し，与えられた条件が成立するか否かを応答CIRによって返してきま

●図……32　条件付き命令

コンディション	ニーモニック	定　義	BSUNビット	式
$00	F	偽		False
$01	EQ	等しい		Z
$02	OGT	オーダでより大きい		$\overline{NAN \vee Z \vee N}$
$03	OGE	オーダでより大きいか等しい		$E \vee (\overline{NAN \vee N})$
$04	OLT	オーダでより小さい		$Z \wedge (\overline{NAN \vee Z})$
$05	OLE	オーダでより小さいか等しい		$Z \vee (N \wedge \overline{NAN})$
$06	OGL	オーダでより大きいか，より小さい		$\overline{NAN \vee Z}$
$07	OR	オーダー		$\overline{NAN}$
$08	UN	アンオーダ		NAN
$09	UEQ	アンオーダ，または等しい		$NAN \vee Z$
$0A	UGT	アンオーダ，またはより大きい		$NAN \vee (\overline{N \vee Z})$
$0B	UGE	アンオーダ，またはより大きいか等しい		$NAN \vee Z \vee \overline{N}$
$0C	ULT	アンオーダ，またはより小さい		$NAN \vee (N \wedge \overline{Z})$
$0D	ULE	アンオーダ，またはより小さいか等しい		$NAN \vee Z \vee N$
$0E	NE	等しくない		$\overline{Z}$
$0F	T	真		True
$10	SF	シグナリング偽	NANコンディションコードがセットされたときにBSUNビットをセットする	False
$11	SEQ	シグナリング等しい		Z
$12	GT	より大きい		$\overline{NAN \vee Z \vee N}$
$13	GE	より大きいか等しい		$Z \vee (\overline{NAN \vee N})$
$14	LT	より小さい		$N \wedge (\overline{NAN \vee Z})$
$15	LE	より小さいか等しい		$Z \vee (\overline{NAN} \wedge N)$
$16	GL	より大きいか，より小さい		$\overline{NAN \vee Z}$
$17	GLE	より大きいか,より小さいまたは等しい		$\overline{NAN}$
$18	NGLE	GLEでない		NAN
$19	NGL	GL　〃		$NAN \vee Z$
$1A	NLE	LE　〃		$NAN \vee (\overline{N \vee Z})$
$1B	NLT	LT　〃		$NAN \vee Z \vee \overline{N}$
$1C	NGE	GE　〃		$NAN \vee (N \wedge \overline{Z})$
$1D	NGT	GT　〃		$NAN \vee Z \vee N$
$1E	SNE	シグナリング等しくない		$\overline{Z}$
$1F	ST	シグナリング真		True

す。コンディションフィールド値が$10以上の命令の場合，68881内部のNAN(NotA Number：無限大÷無限大など，数学的に意味のない演算を行った場合セットされる)ビットがセットされていると，FPSRレジスタのBSUNビットをセットします。

8 サンプルプログラム

68881の使用方法の例として，ROM内データの読み出し，単項演算（SIN(1.0)），二項演算（3.1415+2.7182）の3つのサンプルプログラムをつくってみましたので参考にしてください。

●リスト……1　ROM内データの読み出し

```
/*
 * 68881内部にある円周率データの読み出し
 */

/*  XCの場合には
 *  #define volatile
 *  の1行を入れてください
 */
#include "stdio.h"

union   DAT {
    unsigned char   cdat;
    unsigned short  sdat;
    unsigned int    idat;
    float       fdat;
    double      ddat;
} dat;

struct  CIR {
    unsigned short  response;
    unsigned short  control;
    unsigned short  save;
    unsigned short  restore;
    unsigned short  operation_word;         /* Not used */
    unsigned short  command;
    unsigned short  reservel;
    unsigned short  condition;
    unsigned int    operand;
    unsigned short  register_select;
```

```
    unsigned short  reserve2;
    unsigned int    instruction_address;
    unsigned int    operand_address;        /* Not used */
};

volatile struct CIR *cir = (struct CIR *)0xe9e000;

void main();
void wait_copro();

void main()
{
    SUPER(0);
    cir->command = 0x5c00;      /* FMOVECR #0,FP0   */
    wait_copro(0x0802);
    cir->command = 0x6400;      /* FMOVE.S FP0,xxx  */
    wait_copro(0xb104);
    dat.idat = cir->operand;
    wait_copro(0x0802);
    printf("PAI = %f¥n",dat.fdat);
}

void wait_copro(response)
    unsigned short  response;
{
    unsigned int    i,ack;
    for (i=0; i<0x20; i++) {
        ack = cir->response;
        printf("%04x¥n",ack);
        if ((ack & 0xbfff) == response)
            break;
    }
    printf("**********¥n");
}

/*----実行結果----
 * 0900
 * 0802
 * **********
 * 8900
 * b104
 * **********
```

```
 * 0802
 * **********
 * PAI = 3.141593
 */
```

●リスト……2　単項演算(SIN(1.0))

```
/*
 * sin(1.0)の計算
 */

/*  XCの場合には
 *  #define volatile
 *  の1行を入れてください
 */

#include "stdio.h"

union   DAT {
    unsigned char   cdat;
    unsigned short  sdat;
    unsigned int    idat;
    float       fdat;
    double      ddat;
} dat;

struct  CIR {
    unsigned short  response;
    unsigned short  control;
    unsigned short  save;
    unsigned short  restore;
    unsigned short  operation_word;         /* Not used */
    unsigned short  command;
    unsigned short  reserve1;
    unsigned short  condition;
    unsigned int    operand;
    unsigned short  register_select;
    unsigned short  reserve2;
    unsigned int    instruction_address;
    unsigned int    operand_address;        /* Not used */
};
```

```
volatile struct CIR *cir = (struct CIR *)0xe9e000;

void main();
void wait_copro();

void main()
{
    SUPER(0);
    dat.fdat = 1.0;
    cir->command = 0x440e;       /* FSIN.S  #1.0,FP0 */
    wait_copro(0x9504);
    cir->operand = dat.idat;
    wait_copro(0x0802);

    cir->command = 0x6400;       /* FMOVE.S FP0,xxx  */
    wait_copro(0xb104);
    dat.idat = cir->operand;
    wait_copro(0x0802);

    printf("SIN(1.0) = %f¥n",dat.fdat);
}

void wait_copro(response)
    unsigned short  response;
{
    unsigned int    i,ack;
    for (i=0; i<0x20; i++) {
        ack = cir->response;
        printf("%04x¥n",ack);
        if ((ack & 0xbfff) == response)
            break;
    }
    printf("**********¥n");
}


/*----実行結果----
 * 9504
 * **********
 * 0900
 * 0802
 * **********
```

```
 * 8900
 * b104
 * **********
 * 0802
 * **********
 * SIN(1.0) = 0.841471
 */
```

●リスト……3　二項演算(3.1415+2.7182)

```
/*
 * 3.1415+2.7182の計算
 */

/*  XCの場合には
 *  #define volatile
 *  の1行を入れてください
 */
#include "stdio.h"

union   DAT {
    unsigned char   cdat;
    unsigned short  sdat;
    unsigned int    idat;
    float           fdat;
    double          ddat;
} dat;

struct  CIR {
    unsigned short  response;
    unsigned short  control;
    unsigned short  save;
    unsigned short  restore;
    unsigned short  operation_word;         /* Not used */
    unsigned short  command;
    unsigned short  reserve1;
    unsigned short  condition;
    unsigned int    operand;
    unsigned short  register_select;
    unsigned short  reserve2;
```

```
    unsigned int    instruction_address;
    unsigned int    operand_address;        /* Not used */
};

volatile struct CIR *cir = (struct CIR *)0xe9e000;

void main();
void wait_copro();

void main()
{
    SUPER(0);
    dat.fdat = 3.1415;
    cir->command = 0x4400;       /* FMOVE #3.1415,FP0     */
    wait_copro(0x9504);
    cir->operand = dat.idat;
    wait_copro(0x0802);

    dat.fdat = 2.7182;
    cir->command = 0x4422;       /* FADD.S  #2.7183,FP0  */
    wait_copro(0x9504);
    cir->operand = dat.idat;
    wait_copro(0x0802);

    cir->command = 0x6400;       /* FMOVE.S FP0,xxx   */
    wait_copro(0xb104);
    dat.idat = cir->operand;
    wait_copro(0x0802);
    printf("3.1415 + 2.7182 = %f¥n",dat.fdat);
}

void wait_copro(response)
    unsigned short  response;
{
    unsigned int    i,ack;
    for (i=0; i<0x20; i++) {
        ack = cir->response;
        printf("%04x¥n",ack);
        if ((ack & 0xbfff) == response)
            break;
    }
```

```
    printf("**********¥n");
}

/*----実行結果----
 * 9504
 * **********
 * 0802
 * **********
 * 9504
 * **********
 * 0802
 * **********
 * 8900
 * b104
 * **********
 * 0802
 * **********
 * 3.1415 + 2.7182 = 5.859700
 */
```

RTC

RTCは，現在の日付，時刻を保持するLSIです。X 68000では，RTCを計時動作のほか，指定時刻になると自動的に立ち上がるタイマ動作の実現のために使用しています。ここでは，RTCのアクセス方法などについて説明します。

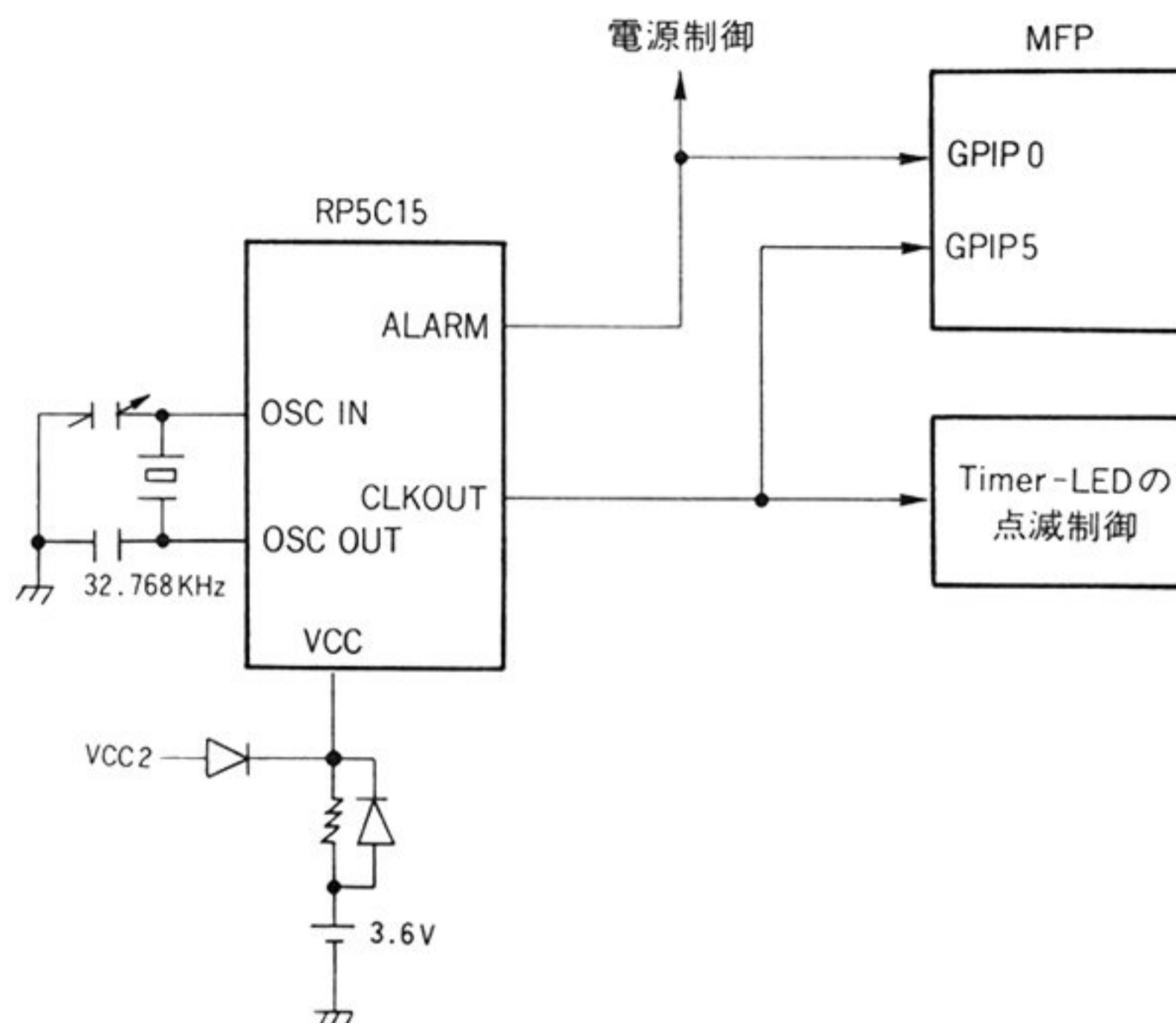
1 RTC周辺ブロック図

X 68000は，日付，時刻などの保持やアラーム（タイマ）動作を行うRTC（リアルタイムクロック）としてリコー製のRP5C15というICを使用しています。X 68000では，このICを時計としての用途のほか，指定時刻での本体電源のON（タイマ機能）や，本体前面にあるTIMER-LEDの点滅制御などに使用しています。

X 68000のRTC周辺回路のブロック図を148ページの図1に示します。基本クロックとしては，時計IC用では一般的な32.768 KHzの水晶発振子を使用しており，この発振周波数を内部で分周して1秒単位のクロックを作成しています。

RP5C15にはALARMとCLKOUTという2つの出力信号があります。ALARM出力は，日付，曜日，時，分があらかじめ設定したものと一致するとLレベルとなる出力です。X 68000ではタイマ機能として使用するとともに，この出力をMFP（マルチファンクションペリフェラル）のGPIP 0のピンに接続し，このピンの出力データが読めるようにしています。

CLKOUT端子は，ソフトウェアによって，Lレベルやハイインピーダンス状態，6種類の周期でのパルス出力のいずれかが選択できるようになっている出力です。X 68000では

●図……1　RTC 周辺ブロック図

TIMER-LED の点滅動作に使用しています。

RP5C15 の電源は，本体背面のメインスイッチが OFF にならないかぎり供給される電源ライン（VCC 2）とバッテリの両方から供給されるようになっています。VCC 2 が供給されているとき，電源は RP5C15 に供給されるとともに，抵抗を通してバッテリを充電しています。VCC 2 が切れたとき（背面のメインスイッチを切ったときや停電したとき）には，この充電していたバッテリによって，時計は動作しつづけます。

●2 RTCのレジスタ

RTCの持つレジスタの一覧を図 2 に示します。RTCには通常 8 ビット（バイト）単位でアクセスしますが，RTCはデータバスを 4 ビットしか持っていないため，有効なのは最下位の 4 ビットだけです。

RTC のレジスタは 2 つのバンク構成になっており，どちらのバンクにアクセスするかは，MODEレジスタ（アドレス：$E8A01B）のビット 0 で指定します。RTCのレジスタのうち，

●図……2　RTC のレジスタ

	BANK 0					BANK 1				
アドレス	内　容	D_3	D_2	D_1	D_0	内　容	D_3	D_2	D_1	D_0
$E8A001	1秒カウンタ					CLKOUT セレクトレジスタ				
$E8A003	10秒　〃					Adjust				
$E8A005	1分　〃					アラーム 1分レジスタ				
$E8A007	10分　〃					アラーム 10分レジスタ				
$E8A009	1時間　〃					アラーム 1時間レジスタ				
$E8A00B	10時間　〃					アラーム 10時間レジスタ				
$E8A00D	曜日　〃					アラーム 曜日レジスタ				
$E8A00F	1日　〃					アラーム 1日レジスタ				
$E8A011	10日　〃					アラーム 10日レジスタ				
$E8A013	1月　〃					(未使用)				
$E8A015	10月　〃					12/24時間 セレクタ				
$E8A017	1年　〃					うるう年 カウンタ				
$E8A019	10年　〃					(未使用)				
$E8A01B	MODE レジスタ	タイマ EN	アラーム EN		BANK 1/0	MODE レジスタ	タイマ EN	アラーム EN		BANK 1/0
$E8A01D	TEST レジスタ	テスト3	テスト2	テスト1	テスト0	TEST レジスタ	テスト3	テスト2	テスト1	テスト0
$E8A01F	RESET コントローラ	1Hz ON	16Hz ON	タイマ RESET	アラーム RESET	RESET コントローラ他	1Hz ON	16Hz ON	タイマ RESET	アラーム RESET

MODEレジスタ，TESTレジスタ，RESETコントローラの各レジスタは，バンク指定はなく，つねに$E8A01B，$E8A01D，$E8A01F番地でアクセスできます。

年月日のレジスタは，すべてBCDフォーマットでアクセスされ，1の位を保持するものと，10の位を保持するレジスタに分かれています。設定する値のレンジチェックなどは行われませんので，ありえない値を設定（1の位を設定するレジスタに$0A以上の値を設定するなど）しないように気をつけてください。

❷·1 CLKOUTセレクトレジスタ

CLKOUTセレクトレジスタのビット配置を図3に示します。CLKOUTレジスタは，CLKOUT端子にどのような信号を出力するのかを決定します。CLKOUTレジスタのうち有効なのは下位3ビットで，これによって図に示すような8種類の出力を選択します。X68000では，CLKOUT端子を本体前面のTIMER-LEDの点滅に使用しており，CLKOUT端子がHレベルのときに点灯するようになっています。このため，このレジスタに'111'を設定すると消灯し，'000'にすると（ハイインピーダンスはHレベルと同じと考えてください）点灯，'101'を設定すると1秒周期で点滅するようになります。

設定を'101'にしたときは立ち上がりエッジ（LからHへの変化）と秒カウンタが進むタイミングが一致しており，'110'に設定したときは立ち上がりエッジが分カウンタが進むタイミングと一致しています。

●図……3　CLKOUTセレクトレジスタ　BANK 1，$E8A001

❷·2 アジャストレジスタ

アジャストレジスタのビット配置を図4に示します。アジャストレジスタは，秒カウンタ(1秒カウンタと10秒カウンタ)をアジャスト，すなわち0にクリアするものです。アジャストはたんなるクリアと異なり，秒カウンタの値が30以上のときには分カウンタがインクリメント

●図……4　アジャストレジスタ　BANK 1, $E8A003

＊秒カウンタが0～29のときにアジャストすると，たんに秒カウンタが0になります。秒カウンタが30～59のときにアジャストすると分カウンタを進めた後，秒カウンタを0にします。

します。たとえば，10時29分29秒のときにアジャストすると10時29分00秒に，10時29分30秒のときにアジャストすると10時30分00秒となります。

アジャストレジスタは最下位ビットだけが有効で，このビットを'1'にするとアジャスト動作になります。

❷·3　12/24時間セレクタ

12/24時間セレクタは，時計を12時間計でカウントするか，24時間計でカウントするかを指定するものです。このレジスタのビット配置を図5に示します。12時間計とした場合，10時間レジスタのビット1が午前/午後を示すビットとなります。'0'で午前，'1'で午後を示すようになります。X 68000では24時間計でカウントを行っています。

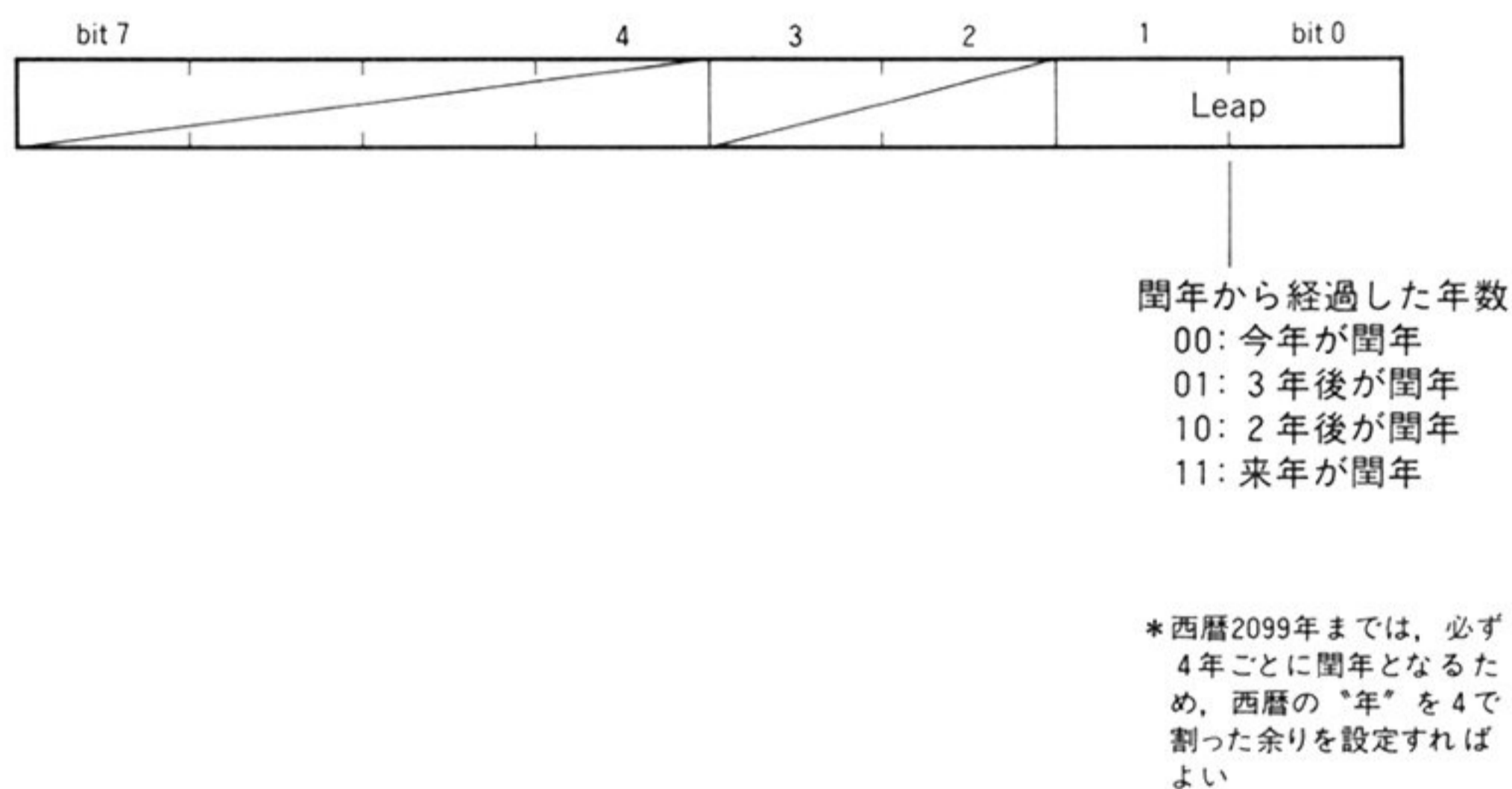
●図……5　12/24時間セレクタ　BANK 1, $E8A015

＊12時間計のとき，10時間カウンタのビット1で午前/午後が示されます。
（0：午前，1：午後）

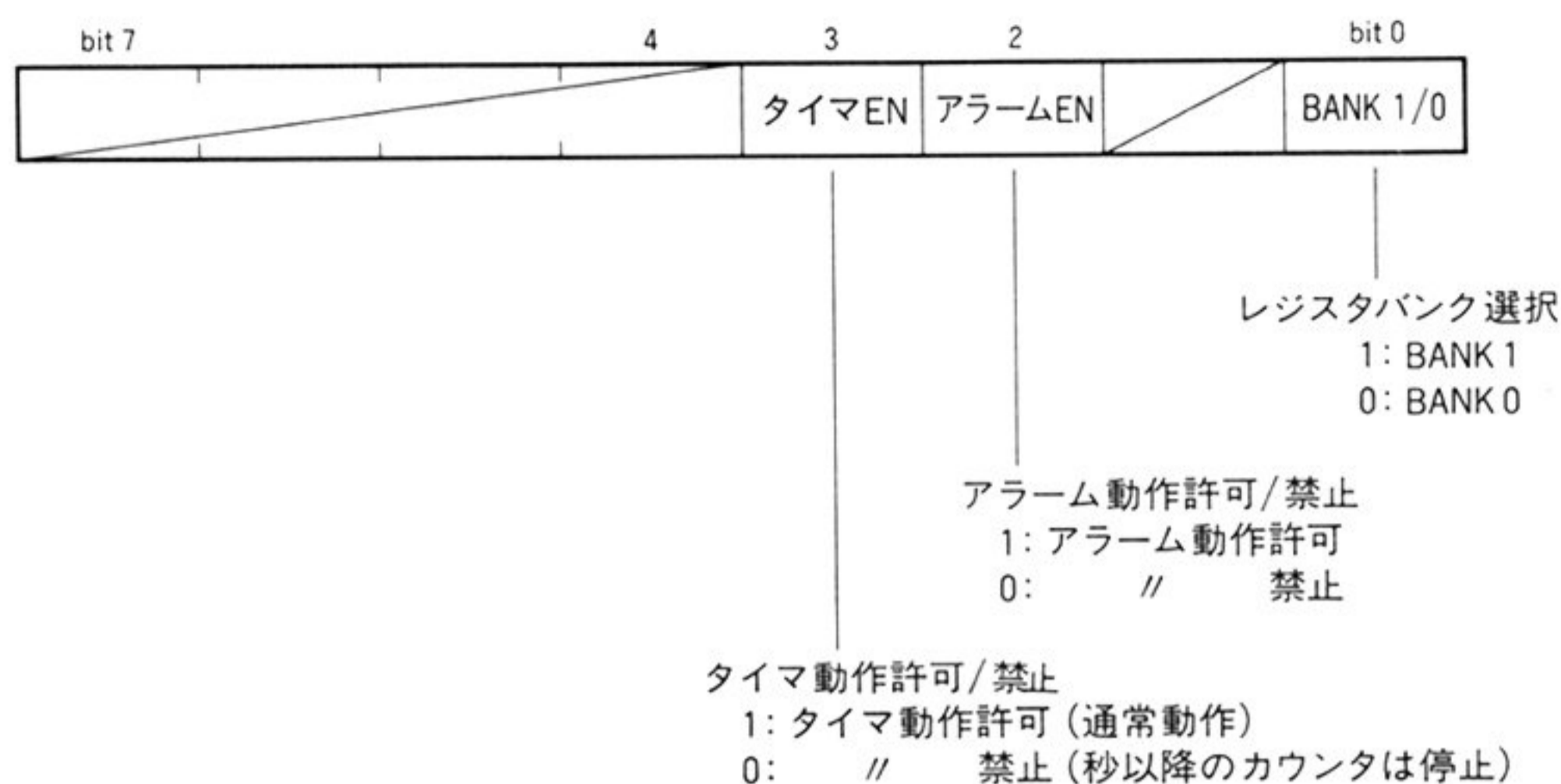
❷·4 閏年カウンタ

ビット配置を図6に示します。閏年(うるうどし)カウンタは，閏年から何年たっているかを設定するレジスタです。'00'のとき，その年が閏年の扱いとなり，2月が29日までカウントされるようになります。閏年カウンタは1年ごとに自動的にインクリメントされますので，閏年の例外(100で割り切れて400で割り切れない年は閏年としない）が発生しないかぎり，そのまま放置しておいても，自動的に閏年の処理が行われます。

この例外措置が次に行われるのは西暦2100年，いまから100年以上も先のことですから(西暦2000年は100で割り切れる年ですが，400でも割り切れるため，閏年となります)，このレジスタには西暦の年数を4で割った余りを書き込めばよいことになります。

●図……6　閏年カウンタ　BANK 1，$E8A017

❷·5 MODEレジスタ

MODEレジスタは，計時動作やアラーム動作の許可/禁止，レジスタバンクの選択を行うレジスタです。ビット配置は図7のようになっています。ビット0は，RP5C15のレジスタバンクのどちら側にアクセスするかを決めるものです。一度書き込むと，次に別の値を書き込むまでその状態のままになります。Human 68 Kは通常動作時にはこのレジスタの変更は行わないようなので，デバッガなどで書き込みを行っても大丈夫です。

●図……7　MODE レジスタ　$E8A01B

ビット 2 はアラーム動作（X 68000 では指定時間に電源が入るタイマ動作用に使用）の許可/禁止制御で，'1' を書き込むとアラーム動作がイネーブルになります。

ビット 3 はタイマ動作（計時動作）の許可/禁止制御で，'0' を書き込むと秒以降のカウント動作が禁止され，'1' を書き込むと通常動作になります。タイマ動作を禁止しても，秒以下のカウンタは動作しており，1 回分のカウントアップは内部で覚えていますので，1 秒以下の時間であれば，このビットを '1' にしたままでも時計がずれることはありません。

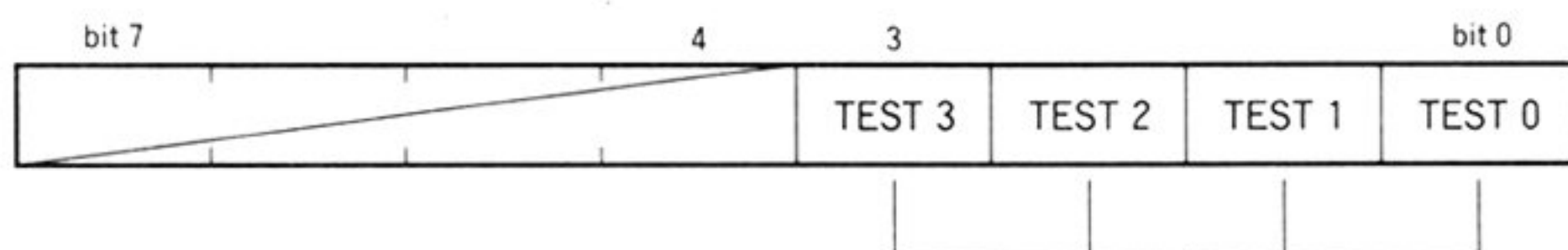
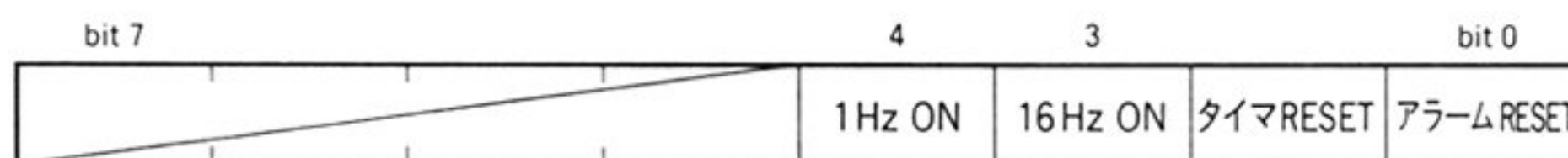
❷·6　テストレジスタ

テストレジスタのビット配置は 154 ページの図 8 のようになっています。これらは RP5C15 のチップメーカ（リコー）での出荷検査用に使うものです（時計が通常よりも速く動いたりするようです)。通常は 0 以外のデータは設定しないでください。

❷·7　RESETコントローラ

RESET コントローラのビット配置を 154 ページの図 9 に示します。RESET コントローラは，アラームの初期化，秒以下のカウンタのリセット，ALARM 出力端子からの出力パルスの選択などを行うレジスタです。

RP5C15 の ALARM 出力端子は，1 Hz のパルス，16 Hz のパルス（いずれもデューティは 50 パーセント)，内部に設定した時刻と現在の時刻の一致の 3 つの要因の OR 条件で出力さ

●図……8　テストレジスタ　$E8A01D

●図……9　RESETコントローラ　$E8A01F

れます。ビット4とビット3は1Hz，16Hzパルスを ALARM 出力端子から出力するか否かを指定するビットで，'0'で出力がON，'1'でOFFになります。両方ともONにすることもできますが，1Hzと16Hzが混ざった波形になってしまうので，実際にはいずれか一方だけをONにするほかないでしょう。X68000ではALARM出力はタイマ動作用として使いますので，これらのビットはいずれも'1'(OFF) に設定します。

タイマリセットは，秒未満の桁（ソフトでは読み出せない部分です）のカウンタを0にクリアするビットです。このビットを'1'にするとクリアされ，'0'にすると通常動作になります。

アラームリセットはアラーム動作の一致検出回路をリセットしますが，このリセットは，少々変わっています。RP5C15のアラーム検出は，日，曜日，時，分の4つの条件の一致を見ていますが，アラームリセットはこれらの比較器を強制的に一致した状態にしてしまい，アラーム時刻（日，曜日，時，分）設定レジスタに書き込みを行うと，その書き込んだレジスタの分だけが不一致状態になります。

このような一見ややこしい動作になっているのは，たとえば，「毎日 18:00 と 22:00」といったような動作を行わせる場合，毎回日付，曜日などを設定しなおす手間を省こうと考えられているためです。アラームリセットによって，新たにアラーム時刻設定レジスタ側に書き込まないかぎり，強制的に一致した状態にされているため，設定動作を省略できます。この例ではアラームの時，分のレジスタだけを変更すればすむわけです。

●3 RTCのアクセス

RTCの時刻はCPUとは関係なく動作していることと，CPU側からは一度には1つのレジスタしかアクセスできないことから，アクセスには少々気を使う必要があります。

❸·1 時刻の読み出し

時刻の読み出しの際，注意しなくてはならないのは，CPUがレジスタを順に読み出している間に桁上がりを起こす可能性があるということです。たとえば，19：59：59 と 20：00：00の境目で，CPU が時計を秒の桁から順に読み出していると，読み出すタイミングによって 20：00：59 となったり，20：59：59 と読み出されたりしてしまうわけです。これを避けるには次のような方法があります。

1)読み出す前に時計を停止させ（MODE レジスタのタイマ EN ビットを使用する），読み終わった後に解除する。
2)時計データを二度読みし，一致しなければもう一度読み出す。
3)1 Hz 信号に同期してデータを読み出す（CLKOUT や ALARM を 1 Hz 出力にして GPIP で読む)。

X 68000 では CLKOUT 端子や Alarm 端子は LEDの点滅やタイマ機能に使用しているため，実際に利用しやすいのは 1)と 2)の方法です。ソフト的にかんたんなのは 1)の方法で，間違って時計を止めたままにしないという点では 2)のほうが安全であるといえるでしょう。

なお，RTC内部の時刻変更タイミングはCLKOUTの立ち上がり（LからHへの変化点）

です。アラーム出力は，CLKOUT と比べ，位相が約 180 度ずれており，アラーム出力の立ち下がりから 96 μs 後に RTC の時刻の更新が行われます。

❸·2 時計データの書き込み

時計データを書き込んでいる最中に桁上がりなどが起こると妙な設定になってしまいます。これを避けるには次のような方法が考えられます。

1)時刻読み出し方法の 1)と同じように時計を止めてから設定する。
2)RESET コントローラレジスタのタイマリセットビットで秒より下の桁をクリアし，停止させてから書き込む。
3)1 Hz 信号に同期させてデータを書き込む。

このうち 3)の方法は，読み出しのときと同じ理由で X 68000 では利用しにくいと思われます。1)の方法では秒より下の桁のカウンタの動作は継続していますので，設定直後の 1 秒の進み方が速くなりますから，2)の方法を併用したほうがよいでしょう。

時計データの書き込みのときには 12 時間計か 24 時間計かの設定，閏年カウンタの設定は必ず行うようにしてください。

❸·3 その他の設定について

以下，ここまでで触れられなかった設定に関する事項をまとめておきましたので参考にしてください。

❸·❸1 年カウンタ

RTC の年カウンタは閏年の処理とは独立して動いているため，設定する年数は西暦の下 2 桁である必要はありません。Human 68 K では西暦から 1980 を引いた値が設定されているものとして扱っています。

❸・❸ 2 曜日カウンタ

曜日カウンタはたんに1日ごとに0～6までの値を順にとっていく7進カウンタで，どの値を日曜日に対応させるかはユーザまかせとなっています。Human 68 Kでは日曜日を0として扱っています。

❸・❸ 3 アラーム機能

アラームの設定は，次のような手順を守るようにしてください。

1)アラームディセーブル（MODEレジスタのビット2を'0'にする）
2)アラームリセット（RESETコントローラレジスタのビット0を'1'にする）
3)100 μs以上ディレイ
4)アラームレジスタへの設定

また，アラーム機能を使う場合には，本体背面のメイン電源スイッチを切らないようにしてください。

●4 サンプルプログラム

時計の読み出しを行うかんたんなサンプルを作成してみましたので，参考にしてください。このプログラムでは二度読み方式を採用しています。

●リスト……1　時計の読み出し

```
/*
 * RTC読み出しサンプル
 */
```

```
/* XCの場合には
 *   #define    volatile
 * の1行を入れてください
 */

#define TRUE    1
#define FALSE   0

char    *dayofweek[7] = {"SUN","MON","TUE","WED","THU","FRI","SAT"};

volatile unsigned char  *rtc_base = (unsigned char  *)0xe8a001;
volatile unsigned char  *rtc_mode = (unsigned char  *)0xe8a01b;

unsigned char   c_time[2][7];

void main();
int  cmp_time();
void read_time();
void print_time();
void bank();

void main()
{
    unsigned int    bnk;
    SUPER(0);
    bank(0);
    bnk = 0;
    read_time(c_time[bnk ^= 1]);
    while(!cmp_time(c_time[0], c_time[1]))
        read_time(c_time[bnk ^= 1]);
    print_time(c_time[0]);
}

int cmp_time(src, dst)
    unsigned char   *src, *dst;
{
    unsigned int    i;
    for (i=0; i<7; i++)
        if (*src++ != *dst++)
            return(FALSE);
    return(TRUE);
```

```
}

void read_time(buf)
    unsigned char   *buf;
{
    volatile unsigned char  *rtc;
    unsigned int    i;
    rtc = rtc_base;
    for (i=0; i<3; i++, rtc += 4) {
        *buf++ = (*rtc & 0xf) + (*(rtc+2) & 0xf)*10;
    }
    *buf++ = *rtc & 0xf;
    rtc += 2;
    for (i=0; i<3; i++, rtc += 4)
        *buf++ = (*rtc & 0xf) + (*(rtc+2) & 0xf)*10;
}

void print_time(buf)
    unsigned char   buf[];
{
    unsigned int    i;
    printf("[YY/MM/DD HH:MM:SS] <%04d/%02d/%02d %02d:%02d:%02d>¥n",
                1980+buf[6],buf[5],buf[4],buf[2],buf[1],buf[0]);
    printf("[Day Of Week       ] <%s>¥n",dayofweek[buf[3]]);
}

void bank(bnk)
    unsigned int    bnk;
{
    if (bnk)
        *(rtc_mode) |= 1;
    else    *(rtc_mode) &= ‾1;
}
```

画面制御

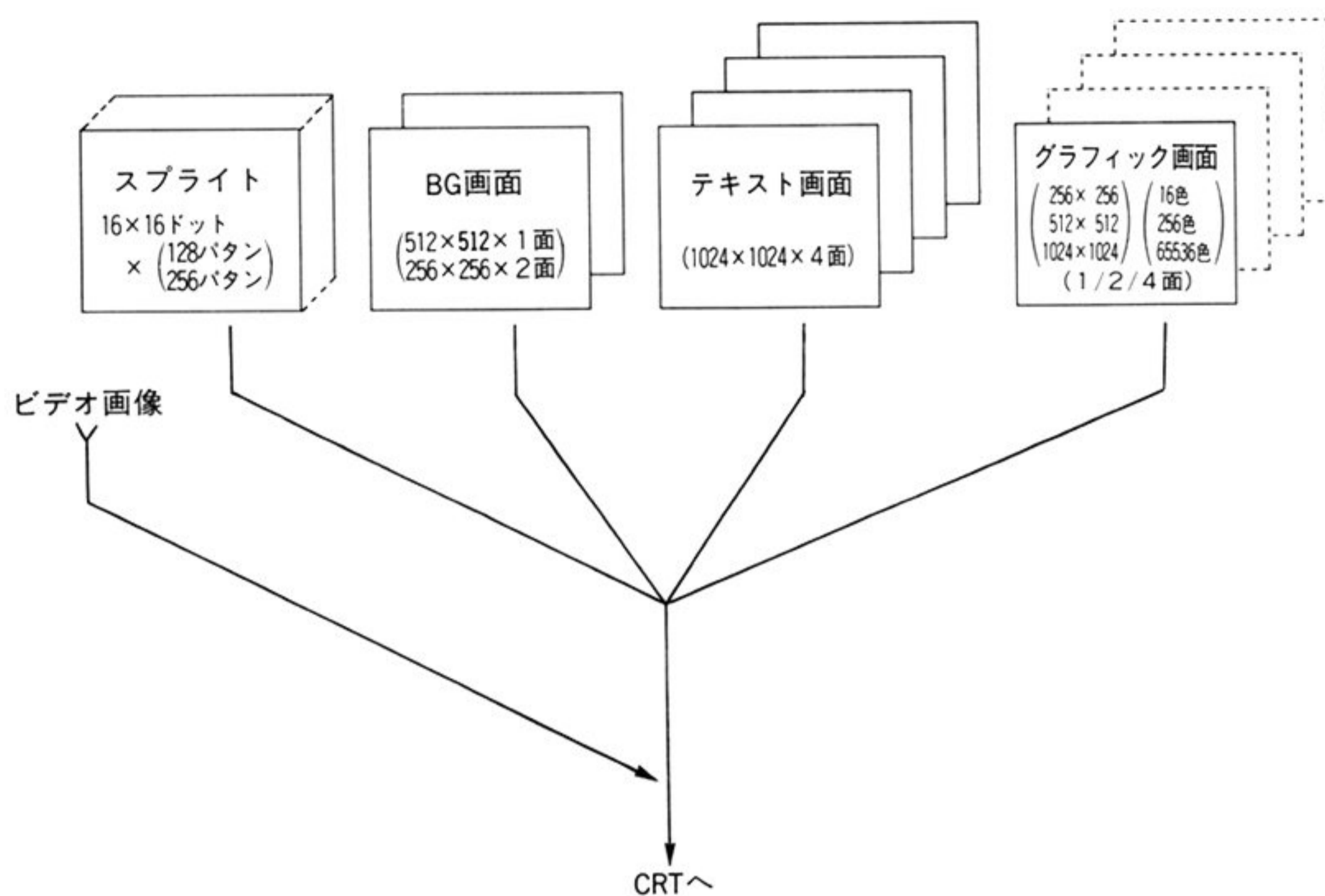
シャープが独自開発したLSI群で固められた画面制御機構は，X 68000のもっとも特徴的な部分であるといえるでしょう。ここでは，X 68000の持つ各種の表示モードや画面制御機構などについて説明します。

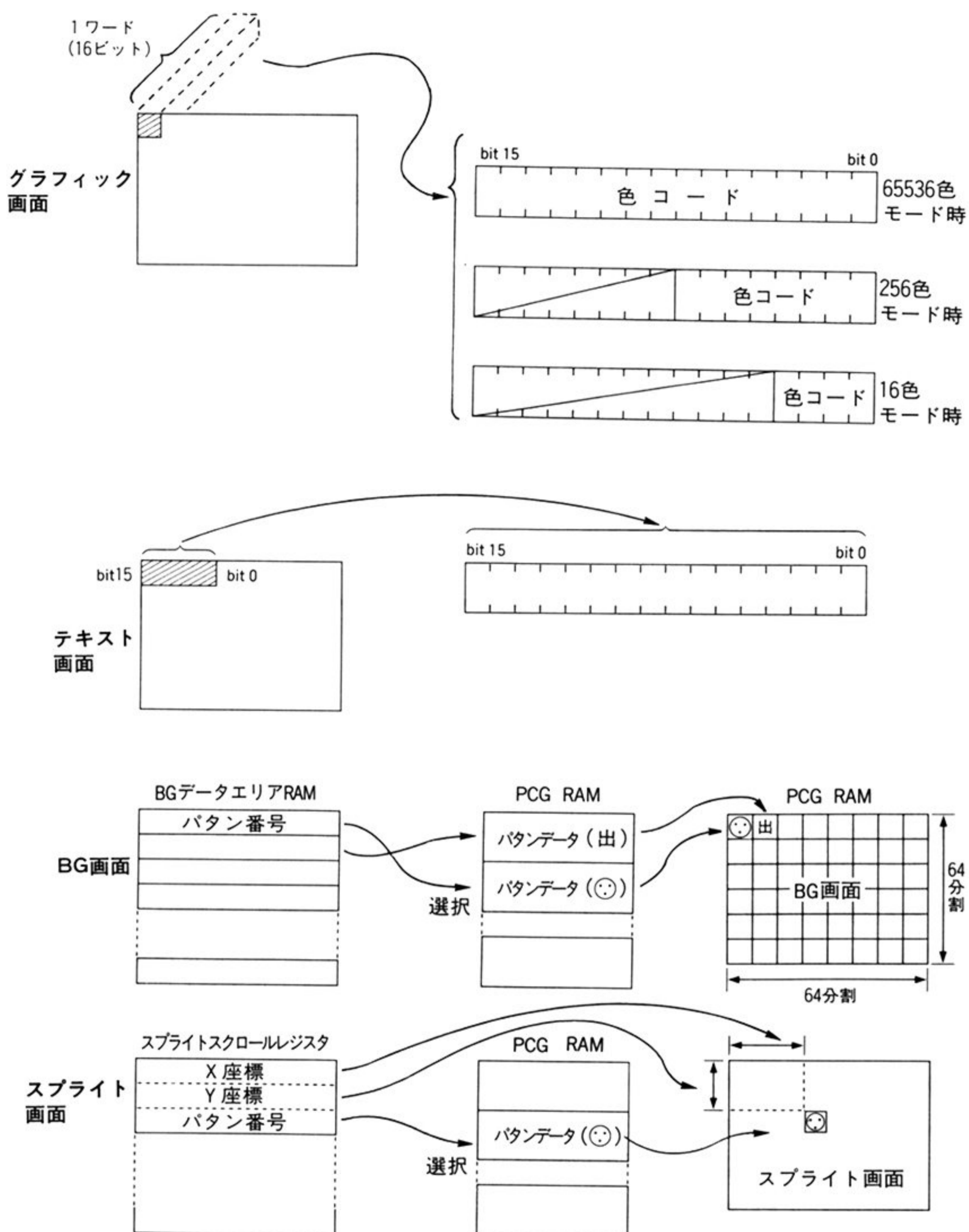
1　X68000の画面構成

X 68000は，他のパソコンには見られないほど強力な画面表示機構を持っています。ビジネス用途におけるパソコンでの画面表示のほとんどは，文字とごくかんたんなグラフ表示程度なので，ハードウェアもそれにあわせ，漢字表示用の画面と，16色程度が扱えるグラフィック画面を持っているだけというのが一般的です。これに対しX 68000は，描画速度が命であるリアルタイムのアクションゲームから，レイトレーシングに代表される，高密度で，数万色以上の画像表示，さまざまな文字フォントにも対応したウィンドウシステムなど，さまざまな「表示」に関する要求に対して，CPUの負荷を極力低減しつつ，柔軟に対応できるような設計が行われています。

X 68000の画面構成を162ページの図1に示します。

X 68000は，グラフィック画面（1～4面），テキスト画面（4面），スプライト（画面上に128個，同一水平線上に32個まで），BG画面(バックグラウンド画面，2面まで)の計4種類の独立した画面を持っており，これらが合成されたうえにビデオ画像との合成（スーパーインポーズ)が行われた後，1つの画面としてCRTに表示されるようになっています。これらの画

●図……1　X 68000の画面構成

面はそれぞれ異なる性格を持っており，目的に応じて使い分けたり，組み合わせて使用することで，多彩な表現を容易に実現できるようになっています。ここでグラフィック，テキスト，BG，スプライトの各画面ごとに，それぞれの構造と特徴などをかんたんにまとめておきましょう。各画面の構造を図2に示しますので，参考にしてください。

●図……2　グラフィック，テキスト，BG，スプライトの各画面の構造概略

❶·1 グラフィック画面

グラフィック画面は，お絵書きソフトやレイトレーシングなど，多くの色を扱いたい場合に適した画面です。カタログなどでうたわれている65536色同時表示が行えるのも，この画面です。モードとしては，65536色モードのほか，256色，16色モードがありますが，どの画面モードでも，つねに画面上の1ドットは1ワード（16ビット）となっています。図に描くとき，データのビット配列が画面に垂直方向になるようにすると説明しやすくなることから，「垂直型」と呼ぶこともあります。指定されたドット位置に対応するメモリ番地に色コードを書き込むだけで，そのドットの色が決まりますし，色コードの取り込みも指定したドット位置のメモリを読み出すだけで行えます。扱う色数が増えても描画に要する手間はまったく変わらず，画面上のデータとの演算もかんたんであるなど，グラフィック表示には有利ですが，半面，1ワードのアクセスでは1ドットしか書き込めないため，文字表示のように多くのドットを同時に書き込むような用途には向いていません。

❶·2 テキスト画面

グラフィック画面とちょうど逆の性格を持っているのがテキスト画面です。テキスト画面という名称から，文字表示しかできないように思われるかもしれませんが，X 68000のテキスト画面は一種のグラフィック画面にほかなりません。任意の位置にドットを打ったり，消したりせるような用途には便利になっています。

テキスト画面が先ほど説明したグラフィック画面と違うのは，テキスト画面はビット配列が水平（横）方向になっているということです。つまり，1ワード分のデータが横方向の16ドットに対応しているわけです。グラフィック画面では16ドットの書き込みをするのには，たとえ白黒表示であっても必ず16回の書き込み動作が必要でしたが，テキスト画面ではこれが1回の書き込みで行えるため，文字パターンのようにあらかじめ用意されているパターンを表示させるような用途には便利になっています。

X 68000では，このようなテキスト画面を4プレーン分持っていて，それぞれのプレーンが色コードの各ビットに対応しています。これによって，最大16色(65536色の中から任意に選択可能）の表示が可能になっています。

❶·3 BG画面

BG (バックグラウンド) 画面は，次に説明するスプライトとともにゲーム向け的な色彩の強い画面です。ゲームの画面ではキャラクタが飛び回るだけではなく，都市や地形図などの背景をともなうのが普通です。このための画面として，先ほどのテキスト画面やグラフィック画面を用いることももちろん可能ですが，ゲームの場合，同じようなパターンが数多く用いられる場合が多いことに注目して，より効率のよい画面制御をめざしたのが BG 画面です。

BG 画面は，全体を縦横とも 64 等分したマス目で構成され，そのうちの 32×32 個分の領域が実際に画面に表示されるようになっています。それぞれのマス目には 1 対 1 に対応したメモリ領域があり，そのパタンの番号 (0～255) を書き込むだけで登録しておいたパタンが表示されるようになっています。スプライトのように各パタンを独立して 1 ドット単位で好きなところに表示するようなことはできませんが，スプライトが画面上最大 128 個までしか表示できないのに対して，BG では 32×32＝1024 個を同時表示 (ただし，使えるパタンは 192 種類まで) できるのが特徴です。

X 68000 は，BG 画面を 2 面まで持てるようになっており，また BG 画面のうちどの部分が画面上に表示されるかを各面独立に 1 ドット単位で指定できるようになっています。これによって背景のスムーズなスクロールが可能になっています。

❶·4 スプライト

スプライトは，定義されたパタンを 1 ドット単位で任意の位置に表示できるものです。X 68000 では縦横がそれぞれ 16 ドットのパターンを 256 個まで定義でき，その中から画面上で最大 128 個 (ただし，同一水平線上には 32 個まで) を同時に表示できます。グラフィック画面やテキスト画面が，グラフィックツールやワープロなど比較的動きの少ない画面を対象としているのに対し，スプライトはアクションゲームなどの，決まった形のキャラクタをすばやく動かすような目的に適したものです。

グラフィック画面やテキスト画面でこのようなゲームをつくろうとすると，キャラクタの移動先にすでにあるデータをあらかじめ読み出しておいて，キャラクタを別の場所に移動させるときにふたたび元に戻すという手間がかかります。複数のキャラクタが重なったときの処理などもなかなか厄介なものです。スプライトを使うと，このような画面上の重なり合いはすべてハードウェアで処理されますので，ソフトウェアはたんにスプライトの表示位置を指定するレジスタに書き込むだけですみ，CPU の負荷は非常に軽くなります。

X 68000の初代機に付属してきたゲーム「グラディウス」などは，このスプライト機能をフルに利用した好例でしょう。

●2 各画面の構成とアドレス配置

X 68000の画面表示回路は，グラフィック，テキスト，BG，スプライトと，性格の異なる4種類の画面を同時に扱いながら，TVとのスーパーインポーズや画像取り込みなどに対応するなど，かなり凝った作りになっています。このため，すべてのレジスタなどを一度に列記すると理解しにくくなると思われますので，ここではまずX 68000の持つ各画面の構造と，表示用メモリのアドレス配置などについて説明していくことにし，画面のON/OFFやプライオリティ制御などの機能については次節以降で説明することにします。

❷·1 グラフィック画面の構成

❷·❶1 グラフィック画面の画面モード

X 68000がサポートを考慮しているグラフィック画面の画面モード一覧を167ページの表1に示します。

X 68000の表示モードは数多くありますが，ドット数に注目すれば，2種類の実画面と，4種類の表示画面の組み合わせになっています。表中，二重丸になっているところは，その画面モードがBASICやXCのライブラリ，IOCSコールなどでサポートされていることを示し，たんなる一重丸になっている画面モードは，システムソフト上のサポートはない（ないし公開されていない)が，XCなどに付属するプログラマーズマニュアルなどでは存在することになっている画面モードであることを示します。

また，表の中で高解像度モード，標準解像度モードという言い方がされていますが，これはたんに水平偏向周波数がそれぞれ31 KHz，15 KHzであることを示しています。X 68000では通常31 KHzモードが使用されていますので，15 KHzモードを標準解像度と呼ぶのは少し

●表……1　X68000のグラフィック画面モード一覧

実画面	表示画面	高解像度モード（水平31kHz）	標準解像度モード（水平15kHz）	色モード×ページ数
1024×1024	768×512	◎	×	16色　1ページ
	512×512	◎	◎(インターレース)	
	512×256	○(二度読み)	○	
	256×256	◎(二度読み)	◎	
512×512	512×512	◎	◎(インターレース)	65536色×1ページ
	512×256	○(二度読み)	○	256色×2ページ
	256×256	◎(二度読み)	◎	16色×4ページ

◎：X-BASICやXCのライブラリ，IOCSからのサポート有
○：IOCS等からのサポートなし，CRTCへの設定は可
×：動作不可

変なことではありますが，この用語はシャープのマニュアル類のあちこちで見かけるので，ここでもその流儀に従うことにしました。

TV放送の水平偏向周波数は15 KHzですので，スーパーインポーズを行う場合には15 KHzモードを使用します。15 KHzモードで設定できる画面モードは，水平方向のドット数が512ないし，256ドットの画面モードだけです。

C O L U M N

インターレースと二度読み

インターレースは，TV放送を行ううえで画面のちらつきを抑えながら，画面データの転送速度を低くするために考えられた方法です。人間の目にぎくしゃくした動きとして見えないようにするには，1/24秒に1枚以上の速度で画面を表示する必要があります。TV放送ではこれを前提に1/30秒に1枚の画面を送っていますが，この速度で画面の表示を行うと，動きは自然に見えるものの，画面全体のちらつきがひどく，非常に見づらくなってしまいます。このため，TV放送では525本ある走査線を偶数番目と奇数番目のものに分け（それぞれの画面をフレームと呼ぶことにします），1/60秒ごとに交互に送ることで画面のちらつきを抑えています。このような表示方式をインターレース方式と呼びます。

X 68000のCRTインタフェースもインターレース方式をサポートしており，15 KHzモード時の512×512ドット表示はインターレース方式で行っています。X 68000のCRTは垂直方向の周波数は60 Hzに固定されています(若干の周波数ずれには追従します)。このため，画面の垂直方向のドット数は31 KHzモードでは512，15 KHzモードでは256が基本となっていますが，偶数番目のフレームと奇数番目のフレームとの区別を行ってCRTに画

面データを送るインターレース方式を使うことで，15 kHz モードでも 512 (＝256×2) ドットの表示が行うことができるようになるわけです。ただし，上下左右の隣りどうしの画素の区別があまり問題とならない TV 画像と異なり，1 ドットずつの区別がなされるパソコンの画像でインターレース表示を行うと，ドットのちらつきがやや目につきます。

二度読みはインターレースとちょうど逆で，31 KHz モードで 256 ドット表示を行うものです。31 KHz モードでは基本的に縦方向は 512 ドットありますが，ある走査線の表示をした後，1 ライン下も同じデータを表示することで縦方向のドット数が半分になったように表示するものです。この方法では各ドットの縦方向の大きさが倍になるため(厳密には15KHz モードのときの 256 ドットモードとは異なりますが)，一応同じ絵が表示できるようになります。

C O L U M N

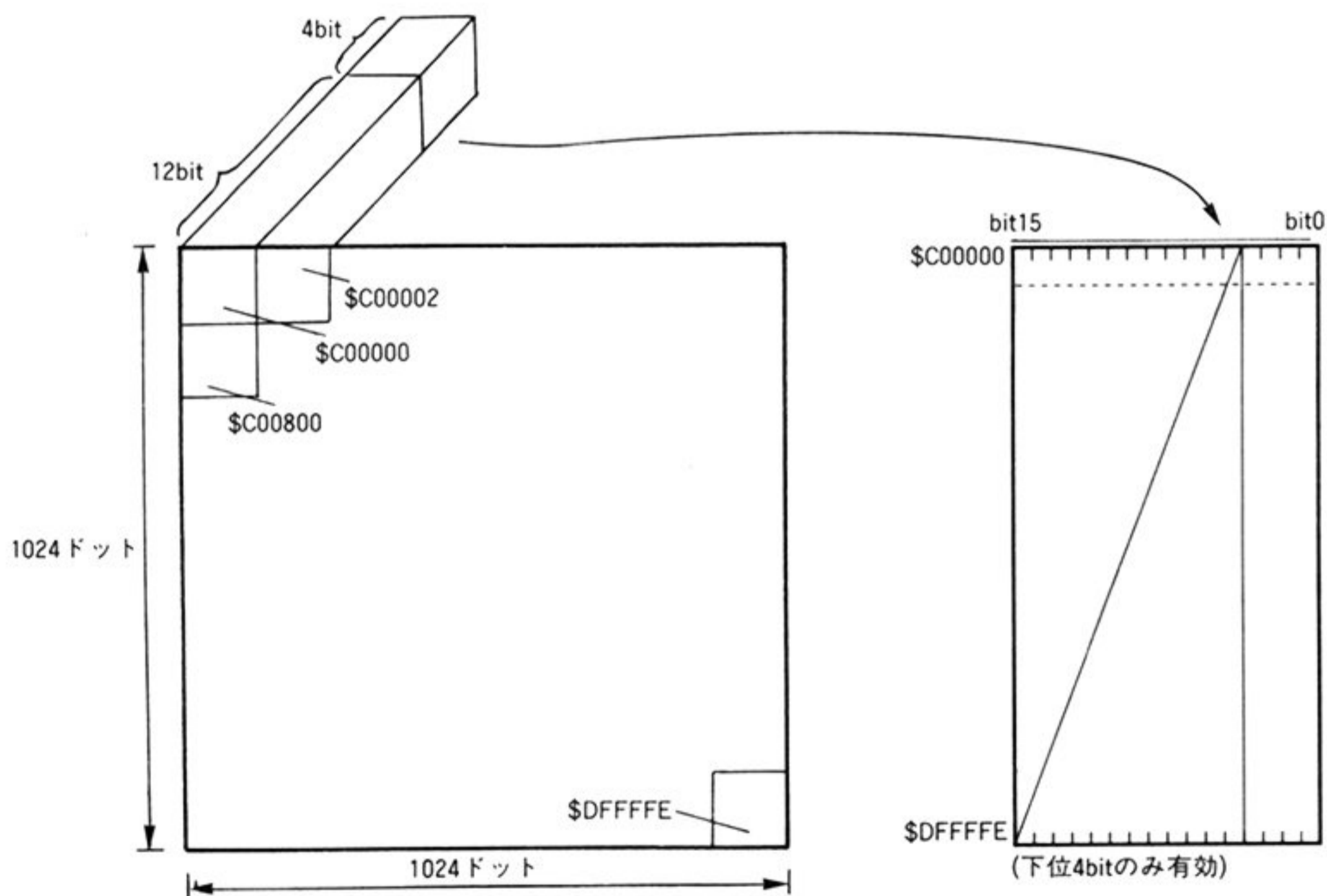
オーバスキャン

インターレースと同じように，オーバスキャンも TV 放送の方式と関係があります。オーバスキャンというのは表示画面の領域を実際の CRT よりも大きくすることで，CRT の表示面全体に画面を表示する方法です。TV 放送の画像は CRT の全面に表示が行われますが，パソコンの画面は通常，表示画面全体が CRT の中央部に表示され，CRT の端には何も表示されない領域が残ります。パソコンの画面が長方形であるのに対して，CRT のほうは丸みを帯びていますし，また CRT の隅のほうはあまり解像度がよくないため，画面全体を見るような用途の多いパソコンでは CRT の中央部を使うようにしているわけです。

X 68000 も 31 KHz モードのときには，このような表示 (アンダスキャンと呼ぶことにします) を行いますが，15 KHz モードのときにはスーパーインポーズでの動作を考慮し，オーバスキャンでの表示が行われます。スーパーインポーズを行ったときに X 68000 の画面のほうがアンダスキャンになっていると，X 68000 側で全面を塗りつぶしたにもかかわらず，画面の端には TV 画面が見えたままになってしまいます。このため，X 68000 の画面表示は 15 KHz モードではオーバスキャン動作にして CRT の表示面全体が扱えるようにしているのです。

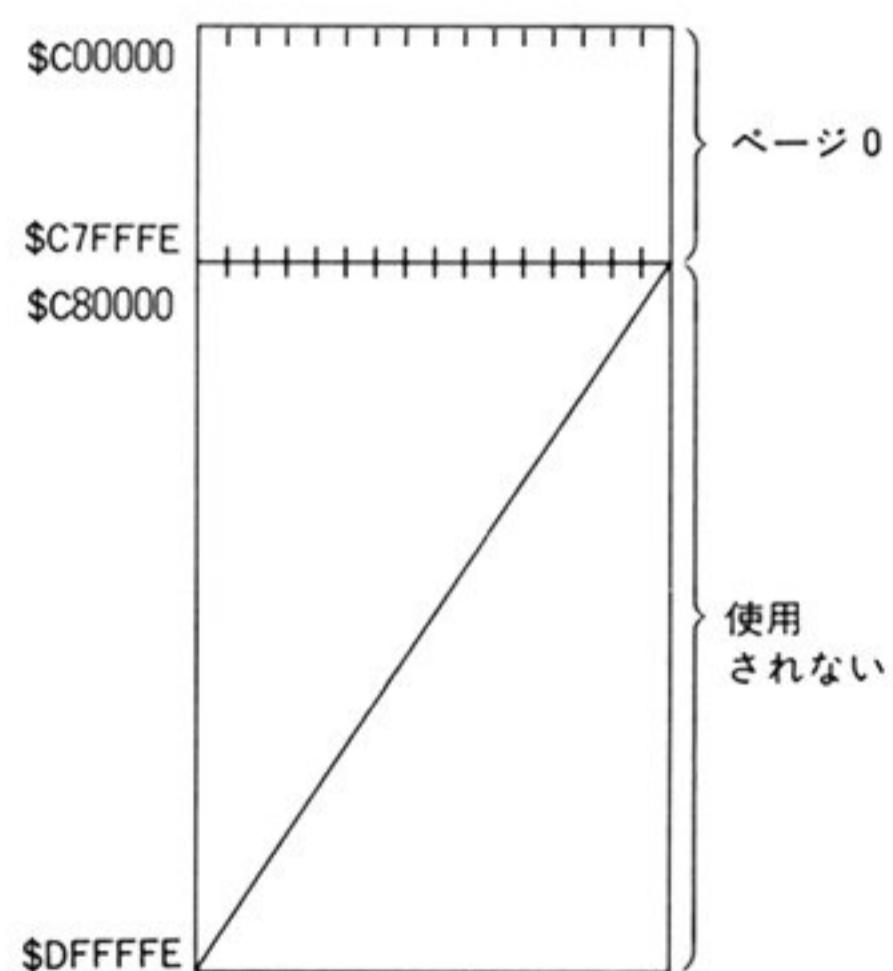
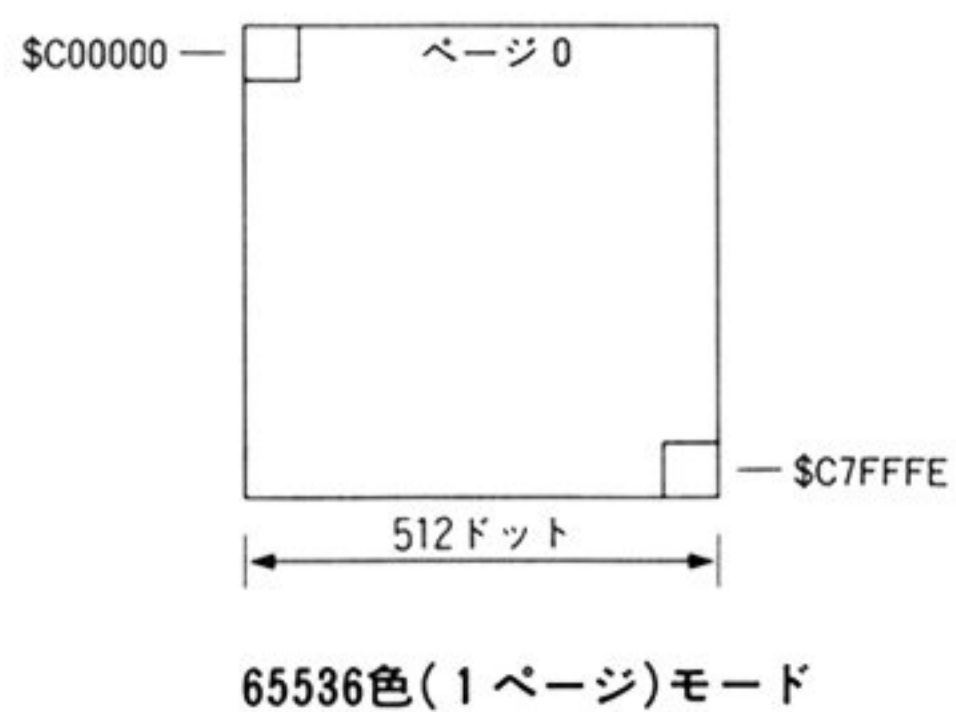
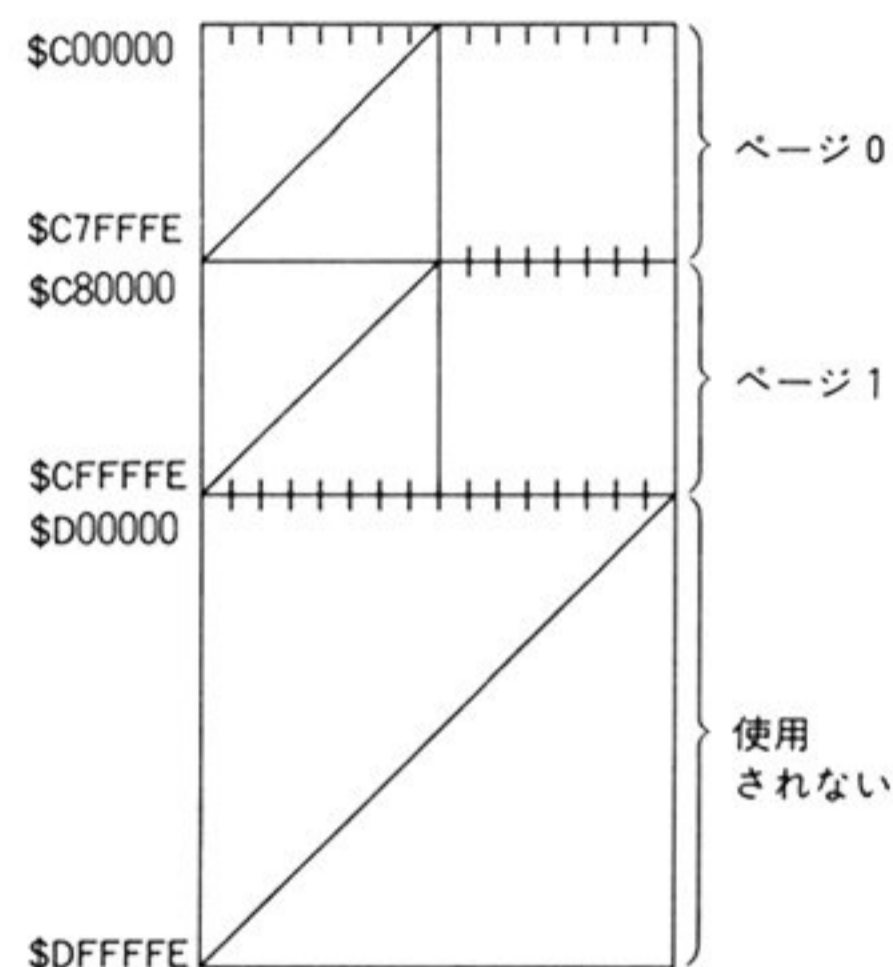
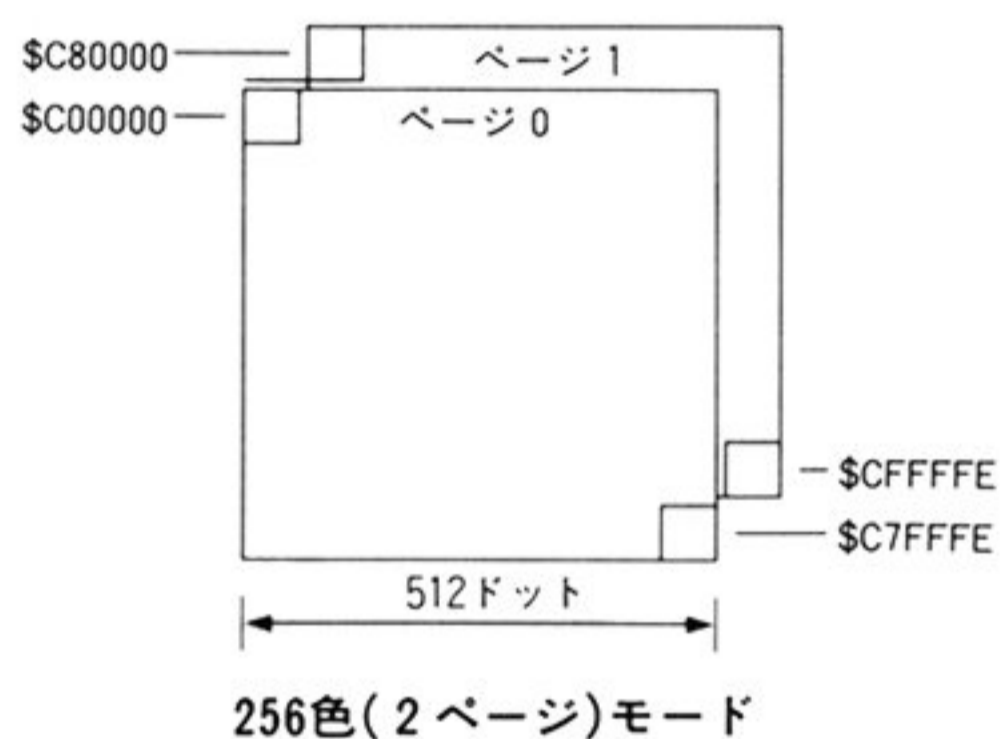
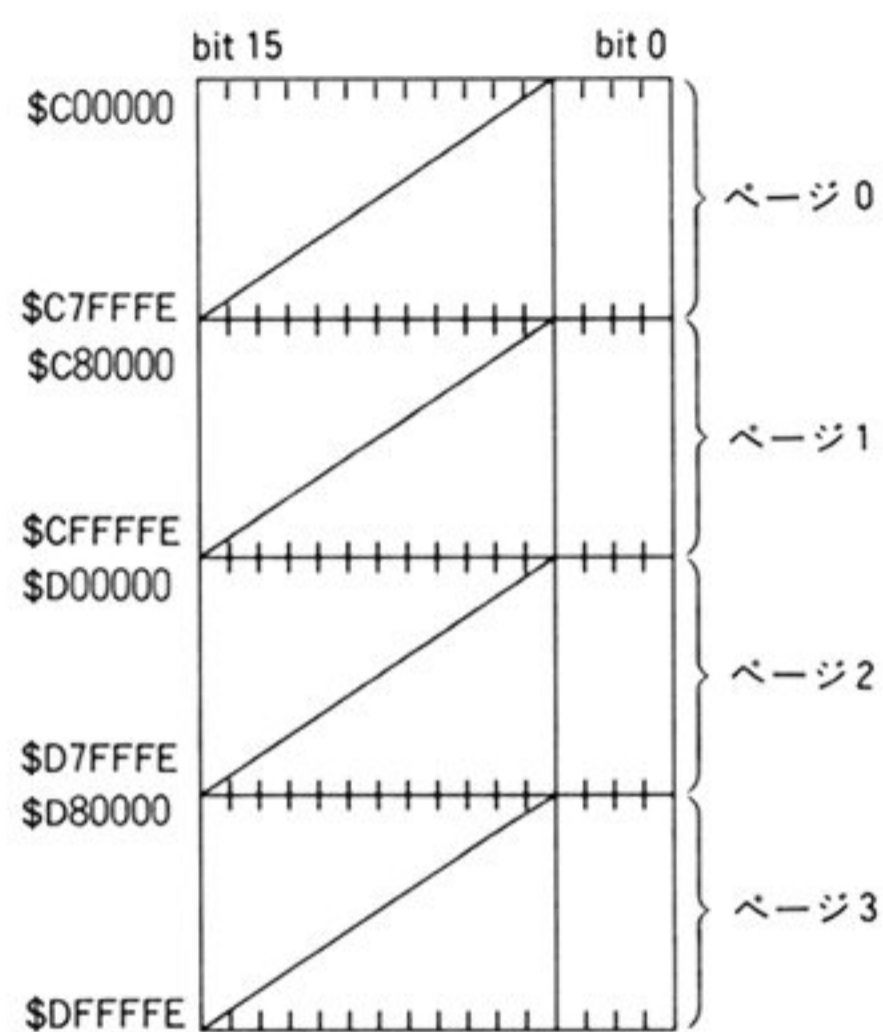
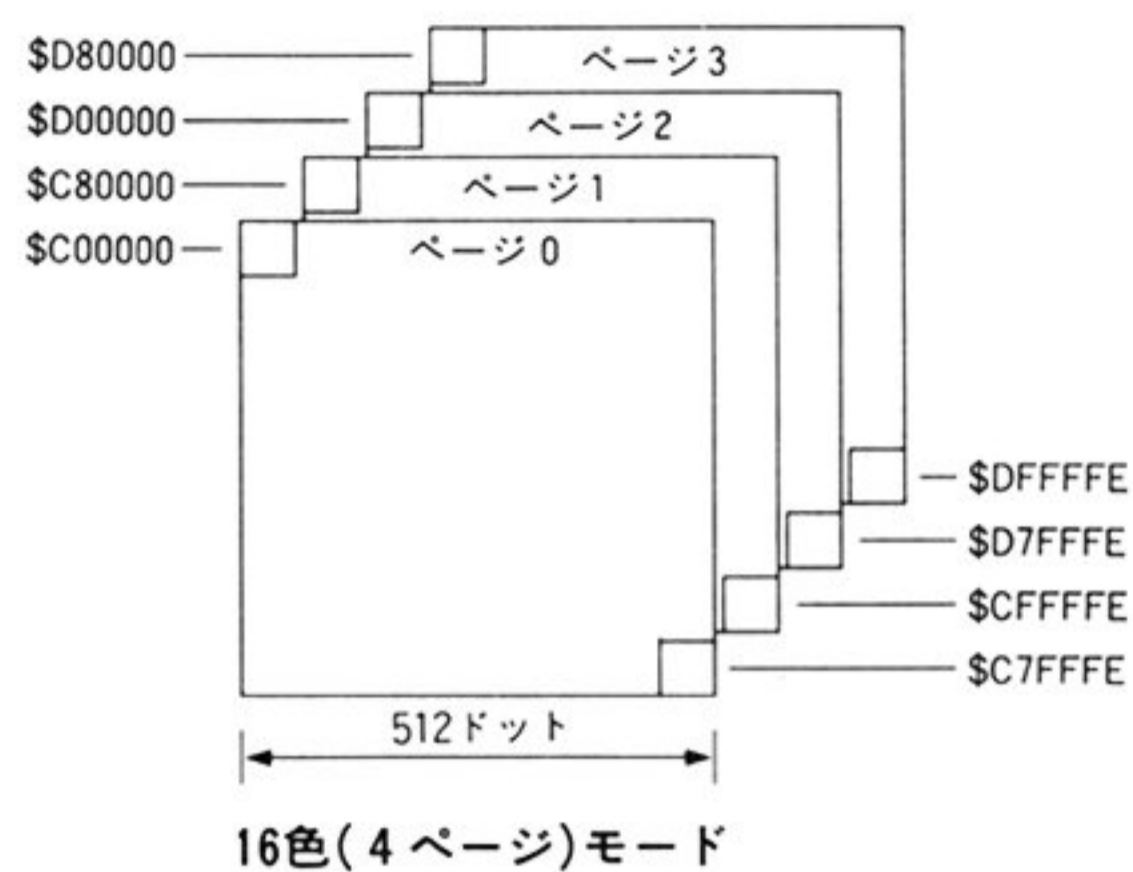
❷·❶2 グラフィックVRAMのアドレス配置

実画面が 1024×1024 ドットのときと，512×512 ドットのときのグラフィック VRAM のアドレス配置を図 3 と図 4 に示します。グラフィック VRAM のアドレス配置は，実画面のモードによって変化しますが，いずれの場合でも，画面上の 1 ドットは 1 ワード（16 ビット）となり，あるドットの右隣りのドットは 2 番地先，さらにその隣りは 4 番地先……というぐあいになります。実画面が 1024×1024 ドットのときは，VRAM の領域は 1 ページで 2 M バイト分の領域を使用し，512×512 ドットのときは各ページが 512 K バイトずつを使用します。つまり，ページ 0 は$C00000～$C7FFFF，ページ 1 が$C80000～$CFFFFF，ページ 2 が$D00000～$D7FFFF，ページ 3 が$D80000～$DFFFFF となります。書き込む色コードは 65536 色モードのときには 1 ワードのデータがそっくりそのまま使われますが，256 色モードのときは下位の 8 ビット分，16 色モード時には下位 4 ビットだけが有効となり，上位ビットは無視されます。

●図……3　グラフィック VRAM のアドレス配置（実画面 1024×1024 ドット時）

●図……4　グラフィック VRAM のアドレス配置（実画面 512×512 ドット時）

16色（4ページ）モード

256色（2ページ）モード

65536色（1ページ）モード

COLUMN

ページとプレーン

ページとプレーンはよく似た概念ですが，本書では表示色やスクロール位置指定などをほかとまったく独立して指定できる単位をページ，テキスト画面のようにほかと組み合わされて色指定を行っているようなものの場合，それぞれの画面をプレーンと呼ぶことにします。

グラフィック画面は，画面モードによって，１つから４つの画面を持つことになります。それぞれの画面は，他のページの画面とは完全に独立して表示制御（色指定，スクロール，プライオリティ，ON/OFF 制御など）が可能であるため，ページと呼びます。

一方，テキスト画面は４つの画面から構成されます。テキスト画面は，この４つの画面のそれぞれが色コードの１ビットに対応しており，４つを使って 16 色（４ビット）のうちのどの色になるかの指定を行うようになっています。また，スクロール位置や ON/OFF 制御なども，４プレーンすべてで連動して扱われます。このため，テキスト画面では，それぞれの画面をプレーンと呼ぶことにしています。

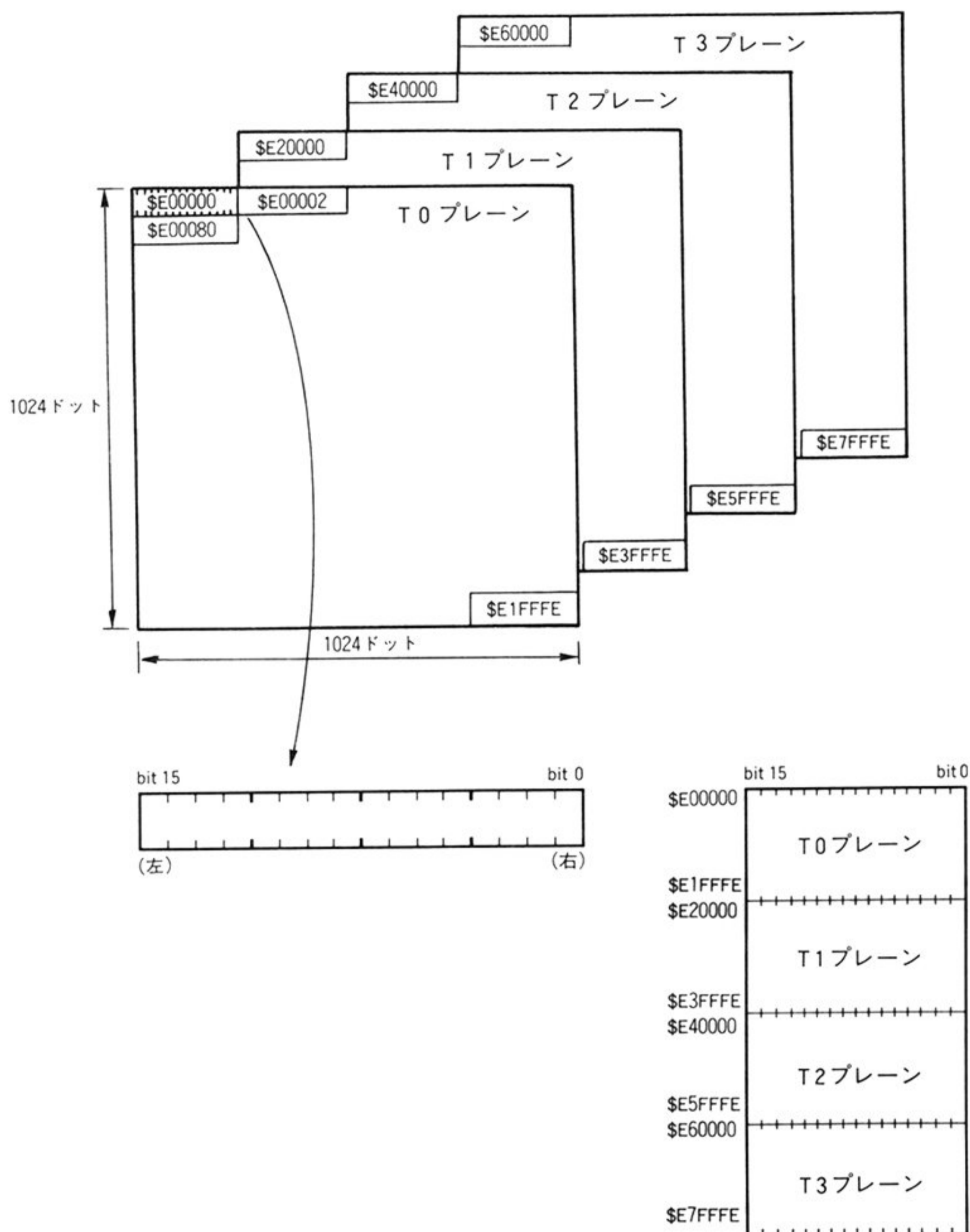
❷·2 テキスト画面の構成

❷·❷1 テキスト画面の画面モード

テキスト画面の画面モードはグラフィック画面と異なり単純です。表示画面サイズはグラフィック画面のサイズに連動しますが，実画面のほうは画面モードによらず，つねに 1024×1024 ドットの大きさがあり，プレーン数は４プレーンとなっています。

❷·❷2 テキストVRAMのアドレス配置

テキスト画面の各プレーンは１ワードが水平方向の 16 ドットに対応するタイプのグラフィック画面です。４つあるプレーンのそれぞれを T０プレーン，T１プレーン，T２プレーン，T3 プレーンと呼ぶことにします。アドレス領域は T0 プレーンが$E00000～$E1FFFF，T1 が$E20000～$E3FFFF，T2 が$E40000～$E5FFFF，そして T3 が$E60000～$E7FFFF となっています（図５）。

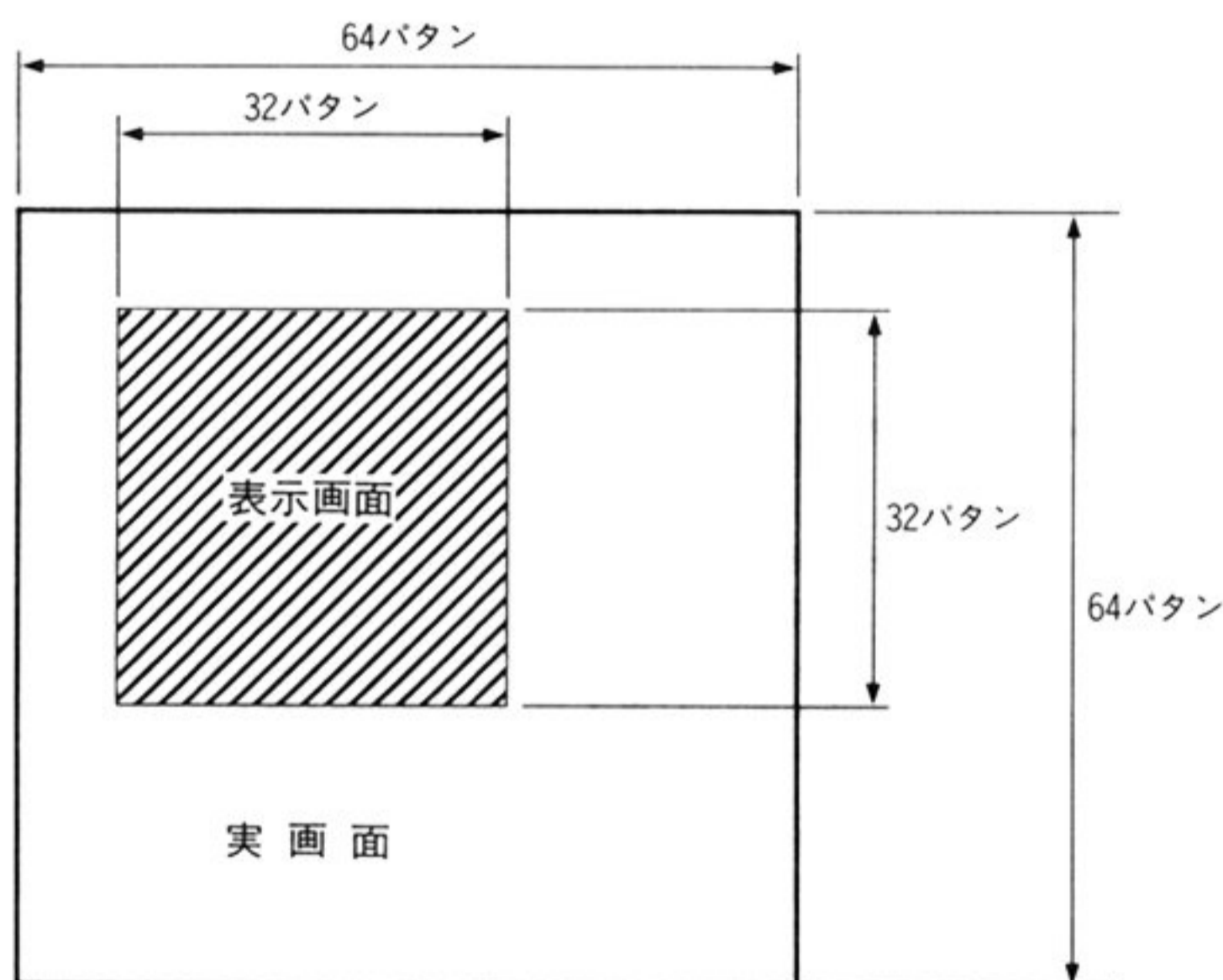
●図……5　テキスト画面のアドレス配置

テキスト画面の色コードは，グラフィック画面のように直接データを書き込むのではなく，T０〜T３の各プレーンの同じ位置に対応するデータによって16色の中から選択されます。あるドットの色コードを知るには4プレーン分（4回）の読み出しが必要であり，やや面倒ですが，書き込みは複数のプレーンに同時に書き込む機能があるため，使用する色数を増やしても，描画速度にはさほど影響しないようになっています。

❷·3 BG画面の構成

❷·❸1 BG画面の画面モード

表示画面が512×512ドットのときには1ページ，256×256ドットモードのときには2ページのBG画面が使用可能です。BG画面の実画面と表示画面の関係を図6に示します。

BG画面の実画面は縦横ともつねに表示画面の2倍になっています。BG画面に使用されるパタンの大きさは，表示画面が512×512ドットモードのときには16×16ドット，256×256ドットモードのときには8×8ドットと変化します。このため，BG画面に並ぶパタンの数は，画面モードによらず，つねに実画面上は64×64個，表示画面上は32×32個になります（BG画面の表示画面サイズは画面モードレジスタ(アドレス$EB 0810)のHRESビット(ビット 0,1)

●図……6　BG画面の実画面と表示画面

画面モードレジスタのHRESビット	表示画面サイズ	実画面サイズ	パタンサイズ(1ヶあたり)
0 0	256×256ドット	512×512ドット	8 × 8 ドット
0 1	512×512ドット	1024×1024ドット	16×16ドット

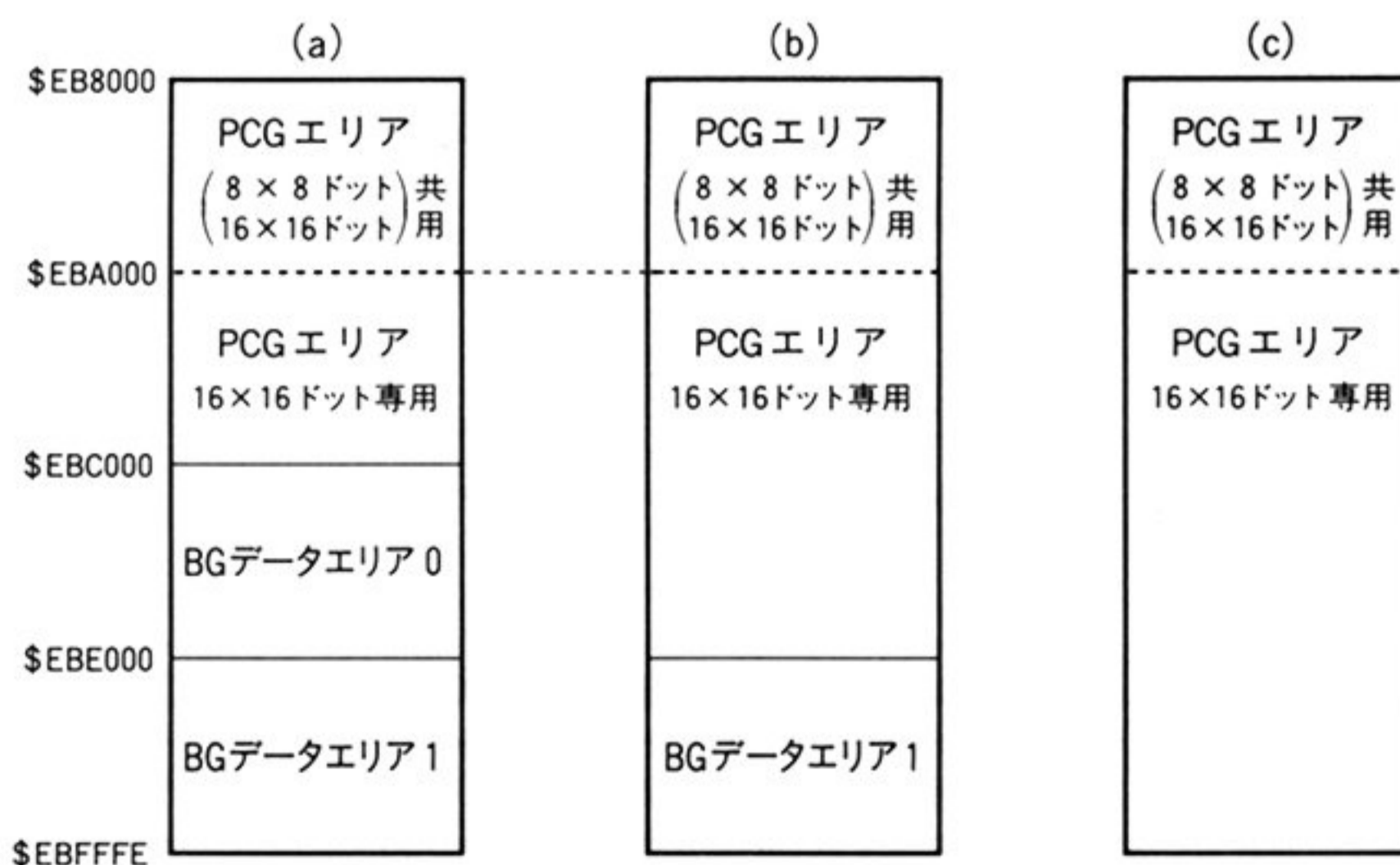
に設定します)。

❷・❸ 2 BG画面用メモリのアドレス配置

BG 画面用のメモリ領域のアドレス配置を図7に示します。BG 画面用の RAM は，表示に利用するパタンを登録する領域（PCG エリア）と，実画面を 64×64（=4096）に分割した各領域と1対1に対応し，どの位置に，どのパタンを表示するかを決める領域(BG データエリア)に分割されます。このうち，PCG エリアはスプライトと共用になっています。

BG 画面用の RAM のうち，前半の 16 K バイト（$EB8000～$EBBFFF）は PCG エリア専用に利用されます。PCG のパタン登録は1ドットあたり4バイト使用されるため，パタンの大きさが 16×16 ドット（画面モードが 512×512 ドット）のときには，パタン1つあたり 128 バイト，8×8 ドット（画面モードが 256×256 ドットモード）のときには1つあたり 32 バイト使用します。パタンが 16×16 ドットのときは，この領域に 128 個定義できることになります。8×8 ドットのときは計算上は 512 個となりますが，BG データエリアが指定するパタン番号が8ビット分しかないため，BG 用に使用可能なのは 256 個分までです。

BG 画面用の RAM の後半 16 K バイトは，8 K バイトずつの領域($EBC000～$EBDFFF, $EBE000～$EBFFFF）に分割されます。BG 画面を2面使うときは両方とも BG データエリアに，1面しか使わないときには前半の 8 K バイトを PCG エリアに，BG 画面を2面とも使わないときは両方とも PCG エリアとしてしまうことができます。

●図……7　PCG エリア，BG データエリアのアドレス配置

これにより，BG画面を1面分しか使わない場合には16×16ドットのパタンを192（＝128＋64）個，BG画面をまったく使わないとき(すべてスプライトパタンとして利用する場合)は256個までのパタンを登録できます。

❷・❸ 3 PCGエリアの構造

PCGエリアのデータ構造を176ページの図8に示します。

PCGエリアのデータは，8×8ドット分(8ロングワード＝32バイト)が定義パタンの単位となっています。画面モードが256×256ドット時のBG画面のように，1パタンが8×8ドットのときには，この1組がそのまま1パタンとして使われ，スプライトや512×512ドットのBG画面のように，1パタンが16×16ドットのときにはPCGデータが4つずつ組み合わされて1パタン分として使われ，番号も1パタン分ごとに取られます。つまり，8×8ドットのときのパタン番号0，1，2，3が16×16ドットのときの0番，4，5，6，7番が1番……というぐあいになるわけです。8×8ドットのパタンがどのように組み合わされるかは図の右下に示しておきました。この例は16×16ドットパタンの番号0のデータが，8×8ドットのときのパタン番号0，1，2，3からどのように構成されるかを示しています。

各PCGの登録データとパタンの対応は図の右側に拡大して示しています。PCGエリアはロングワード（32ビット）単位でアクセスすると，ちょうど横8ドット分のデータが一度に扱えるうえ，ビット配置も最上位の4ビットが左端，最下位の4ビットが右端のドットに対応するようになり，扱いやすいでしょう。

PCGデータは1ドット分が4ビットで表されており，これが各ドットの色コードの下位4ビットになります。上位4ビットはBGデータエリアやスプライトスクロールレジスタにあり，実際に表示されるBGパタンやスプライトごとに指定することができます。この2つが組み合わされることで画面上256色，各パタンごとに16色までの表現が可能となっています。

●図……8　PCGエリアの構造

アドレス	パタン番号 16×16	パタン番号 8×8	PCGエリア
$EB8000	0	0	
$EB8020		1	
$EB8040		2	
$EB8060		3	
$EB8080	1	4	
$EB80A0		5	
$EB80C0		6	
$EB80E0		7	
⋮	⋮	⋮	
$EB9F80	63	252	
$EB9FA0		253	
$EB9FC0		254	
$EB9FE0		255	
$EBA000	64		
$EBA020			
$EBA040			
$EBA060			
⋮	⋮		
$EBBF80	127		
$EBBFA0			
$EBBFC0			
$EBBFE0			
⋮	⋮		
$EBDF80	191		
$EBDFA0			
$EBDFC0			
$EBDFE0			
⋮	⋮		
$EBFF80	255		
$EBFFA0			
$EBFFC0			
$EBFFE0			

PCGエリア専用（$EB8000〜$EBBFE0）
BGデータエリア0を利用（〜$EBDFE0）
BGデータエリア1を利用（〜$EBFFE0）

	bit 31			16	15			bit 0
+$00	P7	P6	P5	P4	P3	P2	P1	P0
+$04	P15	P14	P13	P12	P11	P10	P9	P8
+$08	P23	P22	P21	P20	P19	P18	P17	P16
+$0C	P31	P30	P29	P28	P27	P26	P25	P24
+$10	P39	P38	P37	P36	P35	P34	P33	P32
+$14	P47	P46	P45	P44	P43	P42	P41	P40
+$18	P55	P54	P53	P52	P51	P50	P49	P48
+$1C	P63	P62	P61	P60	P59	P58	P57	P56

8ドット（横）　8ドット（縦）

16×16ドットパタン（パタン番号0）

0	2
1	3

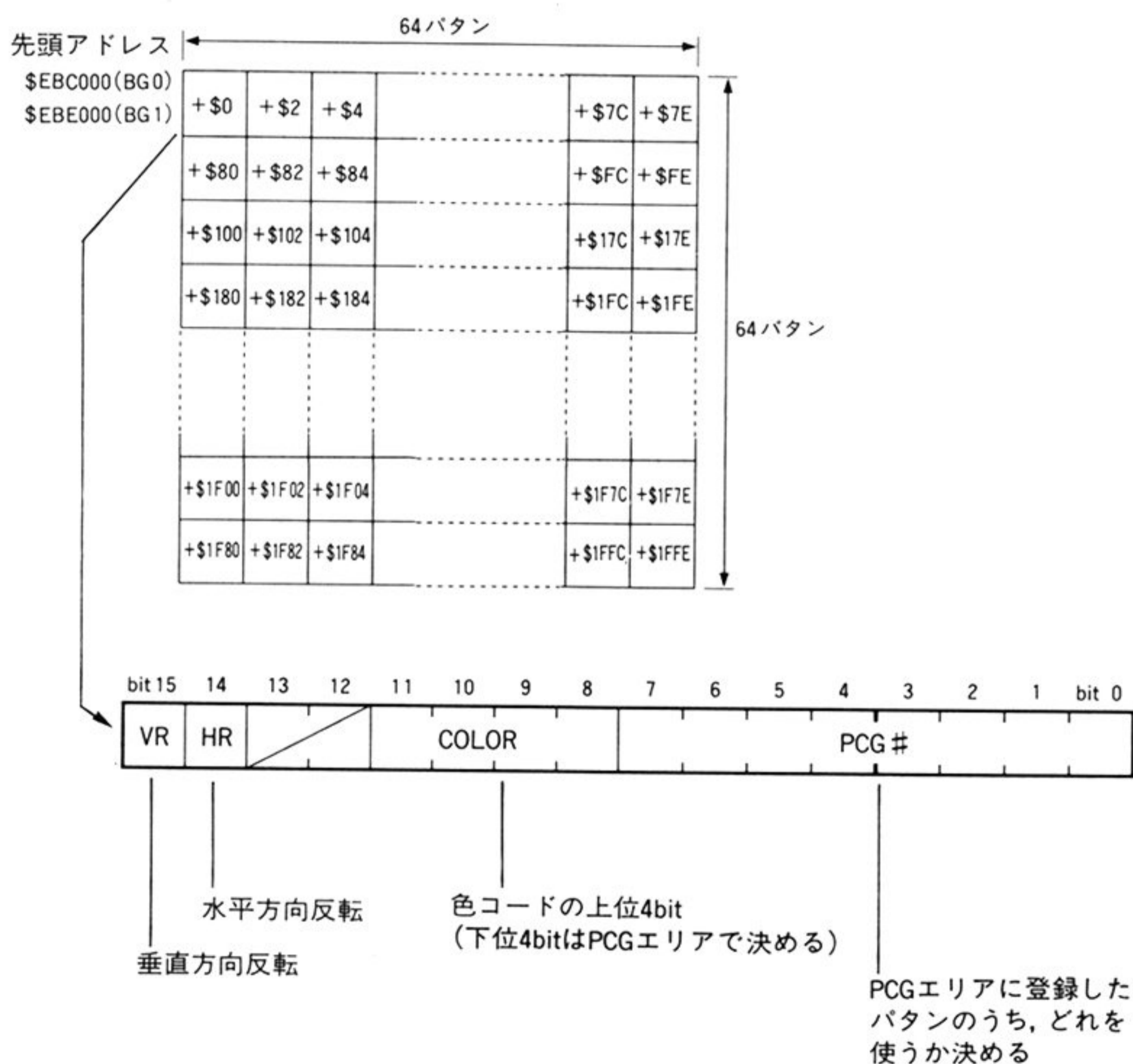
P0　P63

16ドット（縦）　16ドット（横）

❷・❸ 4 BGデータエリアの構造

BGデータエリアの構造を図9に示します。BGの1ブロックあたり1ワードが割り当てられています。下位8ビットはPCGエリアに登録されたパタンの番号，ビット8～11の4ビット（COLOR）は色コードの上位4ビット（下位4ビットはPCGエリアで1ドットごとに指定する）を示します。

ビット15は垂直（上下）方向の反転指定ビット，ビット14は水平（左右）方向の反転指定ビットで，それぞれビットが1になっていると，表示されるパタンの上下方向，左右方向が反転して表示されます。

●図……9　BGデータエリアの構造

❷·4 スプライト画面の構成

❷·❹1 スプライト画面の画面モード

スプライトは，表示画面が512×512ドット，または256×256ドットモードのときに使用可能です。スプライトのパタン登録はBG画面のPCGエリアを共用しますが，BG画面の場合，表示画面サイズによってパタンの大きさが変わるのに対して，スプライトは画面モードによらず，つねに16×16ドットの大きさであるため，BG画面とスプライトのパタン番号が一致しなくなる場合があることに気をつける必要があります。表示画面サイズが512×512ドットモードのときは，BGデータエリアとスプライトで使用するパタン番号は同一になりますが，256×256ドットモードのときには，BGパタン番号が0，1，2，3の4つで表されるパタンがスプライトのパタン番号0に，4，5，6，7の4つがスプライトのパタン番号1番になります。

❷·❹2 スプライト画面のアドレス配置

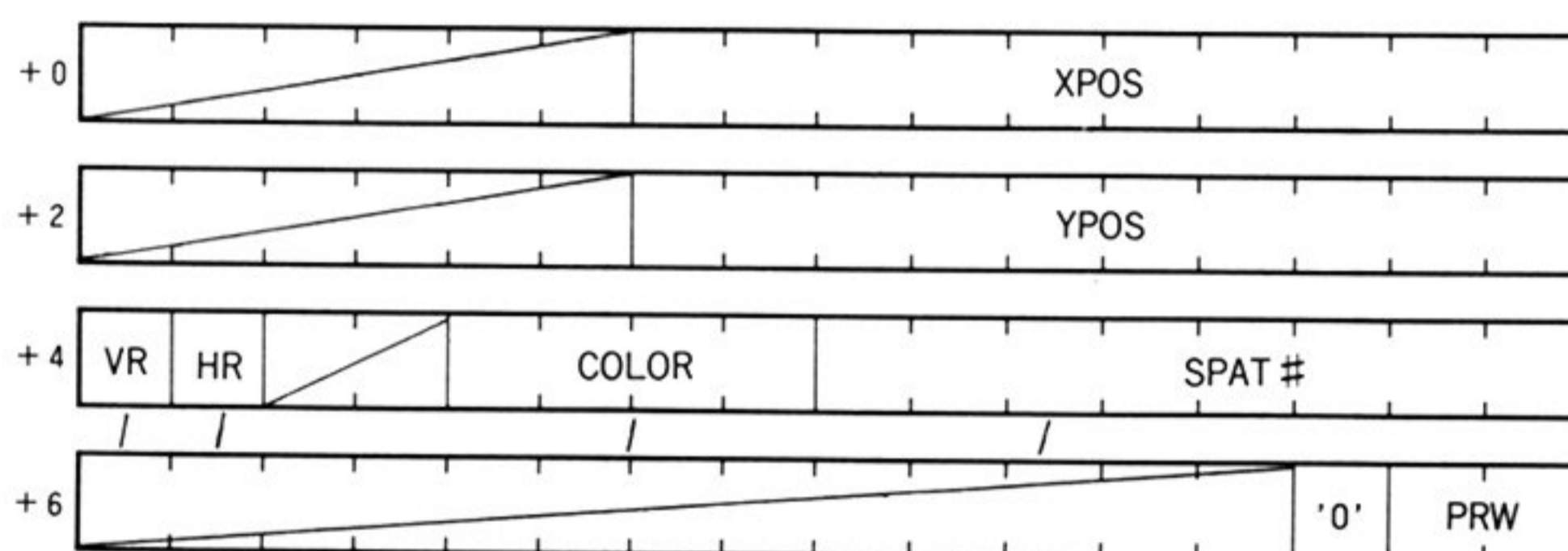
スプライトの制御は，パタンを登録するPCGエリアと，表示場所などを定義するスプライトスクロールレジスタで行います。PCGエリアについては，BG画面のところで説明したので，ここではスプライトスクロールレジスタについて説明することにします。

スプライトスクロールレジスタのアドレス配置とその構造を179ページの図10に示します。

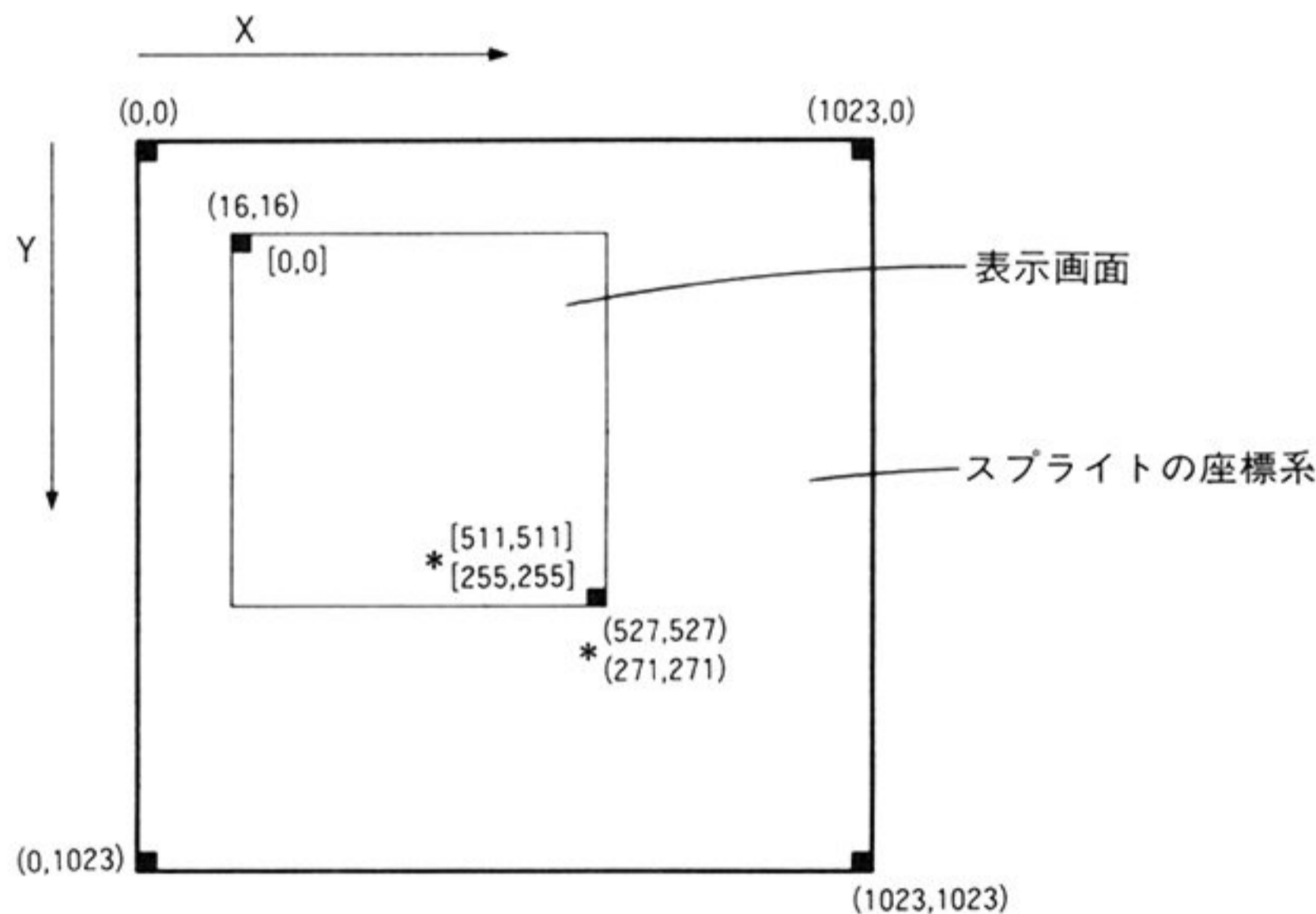
スプライトスクロールレジスタは，表示するパタンの番号，表示位置，表示のON/OFFなどをスプライトごとに指定するもので，1組が8バイト分の領域を使用します。$EB0000～$EB03FFの1Kバイトに計128組用意されていますので，X68000で表示可能なスプライトの数は最大128個になります。ただし，ハード上の制約から，同一水平線上には32個までしか表示できず，33個目以降のスプライトは表示されません。

❷·❹3 スプライトスクロールレジスタの構造

スプライトスクロールレジスタは，スプライト1つあたり4ワード分が割り当てられており，それぞれのレジスタの内容は次のようになっています。

●図……10 スプライトコントローラ スプライトスクロールレジスタ（$EB0000～$EB03FF）

第1ワード，第2ワードは，それぞれスプライトの左上隅の点のX座標，Y座標を指定します。第3ワードはBGデータエリアのデータと同一の構造で，表示されるパタン番号，色コードの上位4ビット，水平/垂直方向の反転指示などを行います。第4ワードは，スプライトとBGの間の表示の優先順位(プライオリティ)を指定するものです。ON/OFF制御やプライオリティ制御用のビットについては次節で説明しますので，ここでは第1，第2ワードだけに注目してください。

スプライトの仮想座標系(実画面)は1024×1024ドット分の領域がありますが，この座標のとり方は，X座標，Y座標とも実際に表示されている画面(実画面)上の座標とは16ドットずつずらしています(図11)。つまり，実画面の左上隅の座標はスプライト画面では(16,16)になります。これは，スプライトを画面の左や上の隅の方向に持っていったときに完全に画面の外に出るまで移動できるようにするためであると思われます。

●図……11　スプライト画面の座標系

●3 画面制御

前節ではX 68000の持つ各種の画面の特徴や，表示用のメモリの構成などについて説明しました。この節では，画面制御ロジックのおおまかな構成について説明した後，各画面どうしの重ね合わせの処理やスクロール，高速クリアや画像取り込みなどの各種の画面制御機構の動作について説明していくことにします。

❸·1 CRTインタフェースの構造

X 68000のCRTインタフェース部のブロック図を図12に示します。X 68000の画面機構は，CRTコントローラ，ビデオコントローラ，スプライトコントローラの3種類のLSIによって実現されています。これらのLSIは，すべてシャープがX 68000用に開発したものです。各コントローラのおおまかな役割分担は次のようになっています。

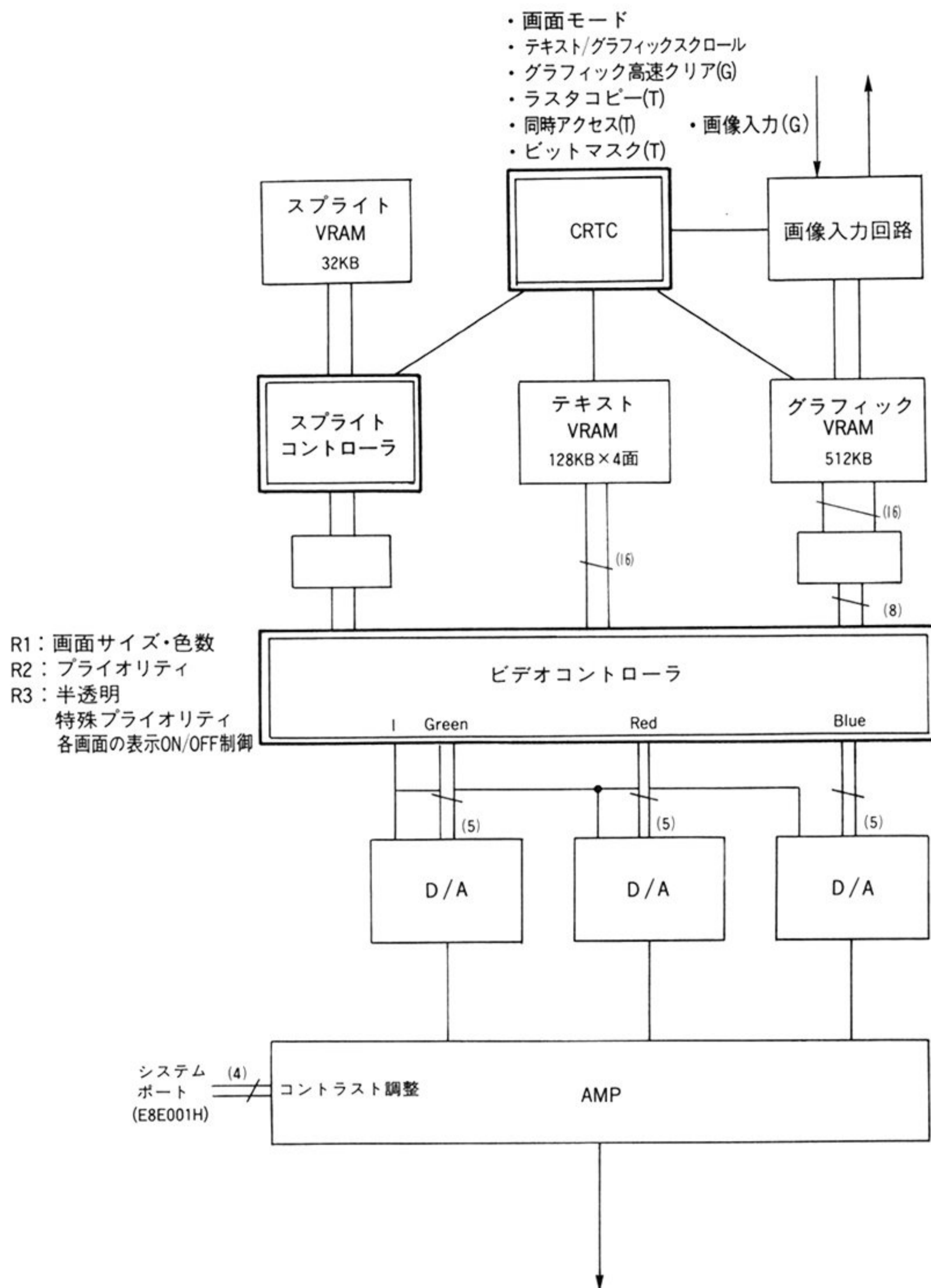
❸·❶1 CRTC

CRTコントローラ(CRTC)は，CRTインタフェース全体が動作するために必要な各種タイミング信号の発生とテキスト画面，グラフィック画面の制御がおもな仕事です。テキスト画面やグラフィック画面のスクロール処理，高速クリアや画像取り込み回路のコントロールなども，CRTCが行っています。

CRTCの持っているレジスタの一覧を183ページの図13に示します。このうち，R 00～R 08はCRTディスプレイとのタイミング調整用，R 09～R 19とR 21，R 23，CRTC動作ポートは画面スクロールや高速クリア動作，画像取り込みなどの制御，R 20は画面モードの設定に使用されます。

❸·❶2 ビデオコントローラ

ビデオコントローラは，グラフィックVRAMやテキストVRAMのデータ，スプライトコ

●図……12　CRTインタフェースブロック図

●図……13　CRTC内部レジスタ一覧

			bit15 … bit0
水平タイミング制御	R00	$E80000	水平トータル
	R01	$E80002	水平同期終了位置
	R02	$E80004	水平表示開始位置
	R03	$E80006	水平表示終了位置
垂直タイミング制御	R04	$E80008	垂直トータル
	R05	$E8000A	垂直同期終了位置
	R06	$E8000C	垂直表示開始位置
	R07	$E8000E	垂置表示終了位置
水平位置微調整	R08	$E80010	外部同期水平アジャスト
ラスタ割り込み用	R09	$E80012	ラスタ番号
テキスト画面スクロール	R10	$E80014	X位置
	R11	$E80016	Y位置
グラフィック画面スクロール	R12	$E80018	X0
	R13	$E8001A	Y0
	R14	$E8001C	X1
	R15	$E8001E	Y1
	R16	$E80020	X2
	R17	$E80022	Y2
	R18	$E80024	X3
	R19	$E80026	Y3
メモリモード/表示モード制御	R20	$E80028	SIZ / COL / HF / VD / HD
同時アクセス/ラスタコピー/高速クリアプレーン選択	R21	$E8002A	MEN / SA / AP / CP
ラスタコピー動作用	R22	$E8002C	ソーラスタ / ディスティネーションラスタ
テキスト画面アクセスマスクパタン	R23	$E8002E	マスクパタン
画像取り込み/高速クリア/ラスタコピー制御	CRTC動作ポート	$E80480	RC / '0' / FC / VI

●図……14　ビデオコントローラレジスタ一覧

	未使用	実画面サイズ	色モード
R0 ($E82400)		SIZ	COL

	未使用	画面間プライオリティ制御			グラフィック画面間プライオリティ制御			
		スプライト	テキスト	グラフィック				
R1 ($E82500)		SP	TX	GR	GP3	GP2	GP1	GP0

	ビデオカット	半透明/特殊プライオリティ制御							つねに'0'	画面ON/OFF制御						
										スプライト	テキスト	グラフィック				
R2 ($E82600)	YS	AH	VHT	EXON	H/P	B/P	G/G	G/T	'0'	SON	TON	GS4	GS3	GS2	GS1	GS0

ントローラの出力などをもとに，各画面のON/OFFや画面間のプライオリティ処理，半透明処理やカラーパレットの処理などを行っています。

ビデオコントローラの持つレジスタ一覧を図14に示します。

R0は画面モードの設定，R1はプライオリティ制御，R2は画面のON/OFFや特殊プライオリティなどの制御に使用されます。カラーパレットもハード的にはビデオコントローラに含まれているのですが，プログラム上からはまったく異質なものであるため，ここではビデオコントローラのレジスタには含めず，後でまとめて説明することにします。

❸·❶3　スプライトコントローラ

スプライトコントローラはスプライト画面とBG画面の表示制御を行います。BG画面のスクロールやスプライトの表示位置の設定，個々のスプライトとBG画面間のプライオリティの制御もスプライトコントローラが行っています。

スプライトコントローラのレジスタ一覧を185ページの図15に示します。各スプライトと1対1に対応し，表示位置やパターン番号，色コードの上位4ビットなどを保持するスプライトスクロールレジスタが128組(1組は4ワード)，BG画面の表示位置やON/OFF制御などを行うBGスクロールレジスタが5ワード，スプライト/BG画面の画面モード制御を行うレジスタが4ワード分あります。

X68000の画面表示は，これらの協調動作によって行われているため，画面モードの設定など，各コントローラの間でおたがいにつじつまをあわせておかなくてはならないものについては各コントローラごとに設定するレジスタを持っています。コントローラのレジスタを直接操作する場合，他のコントローラの設定も変更する必要がないかどうかを，考慮しておく必要が

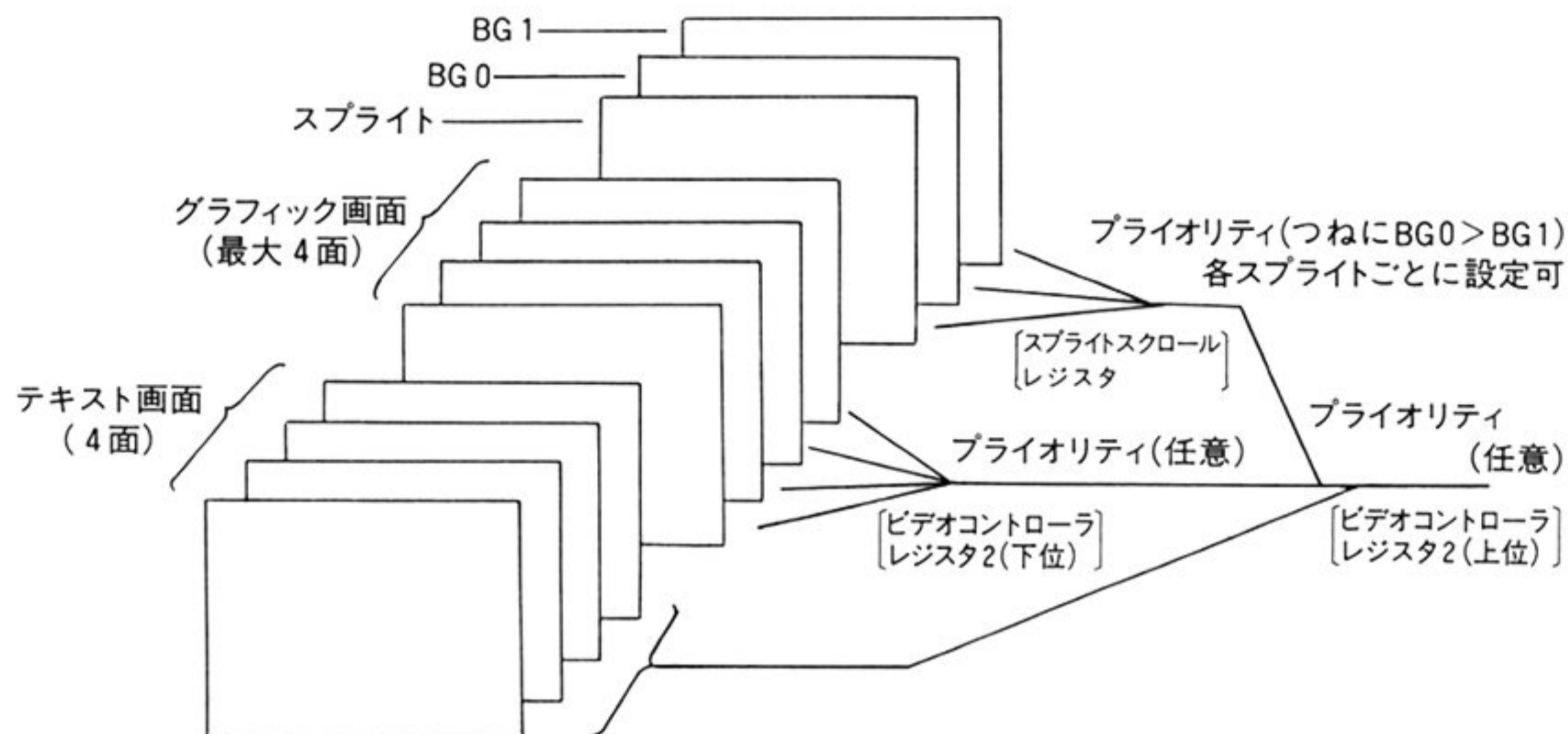
●図……15　スプライトコントロールレジスタ一覧

分類	名称	アドレス
スプライトスクロールレジスタ	スプライト＃0	$EB0000
		$EB0002
		$EB0004
		$EB0006
	：	：
	スプライト＃127	$EB03F8
		$EB03FA
		$EB03FC
		$EB03FE
BGスクロールレジスタ	BG0	$EB0800
		$EB0802
	BG1	$EB0804
		$EB0806
	BGコントロール	$EB0808
画面モードレジスタ	水平トータル	$EB080A
	水平表示位置	$EB080C
	垂直表示位置	$EB080E
	解像度設定	$EB0810

あります。

❸・2　画面のON/OFF，プライオリティ制御機構

X 68000の画面のON/OFFやプライオリティの制御構造を186ページの図16に示します。スプライトとBG画面間，グラフィック画面の各ページ間でのプライオリティ制御や各々

●図……16　プライオリティ制御

の ON/OFF 制御が行われた後，グラフィック画面，テキスト画面，スプライト＋BG の各画面間のプライオリティ制御，ON/OFF 制御が行われます。

スプライトコントローラでは BG 画面とスプライトの制御を行います。BG 画面は BG コントロールレジスタによって 2 画面独立に ON/OFF 制御ができ，スプライトはスプライトスクロールレジスタのプライオリティ制御ビットによって 1 つずつ独立して表示 ON/OFF と BG 画面との間でのプライオリティの制御ができます。

ビデオコントローラは，グラフィック画面のページ間のプライオリティ制御と ON/OFF 制御に加え，グラフィック，テキスト，スプライト＋BG の各画面のプライオリティや ON/OFF の制御を行っています。

＊　スプライトと BG 画面は，スプライトコントローラで合成された後にビデオコントローラに送り込まれますので，ビデオコントローラではスプライトと BG 画面の区別は行えず，レジスタの名称などではたんにスプライトとして扱われています。

❸・❷1　ビデオコントローラによるON/OFF，プライオリティ制御

ビデオコントローラの持つレジスタのうち，画面のプライオリティ制御に関するレジスタは R 1，画面の ON/OFF に関係するレジスタは R 2 の下位 8 ビットです。R 2 の上位 8 ビットは特殊プライオリティや半透明の制御用に使用します。ビデオコントローラのレジスタはすべて READ/WRITE 可能なので，現在の設定をいったん読み出した後，必要なビットだけを書き換えることが可能です。

1 プライオリティ設定

ビデオコントローラのＲ１のビット配置を図17に示します。ビデオコントローラのＲ１は，下位８ビットがグラフィック画面のページ間のプライオリティの指定，上位８ビットはグラフィック，テキスト，スプライト＋BGの各画面のプライオリティの指定用となっています。

2 グラフィック画面のページ間プライオリティ

グラフィック画面のプライオリティ設定は，もっともプライオリティの高いページ番号をビット０と１で，次のプライオリティのページ番号をビット２，３で，３番目をビット４，５で，もっともプライオリティの低いページ番号をビット６，７で指定するようになっています。異なるプライオリティのところに同じページ番号を指定することは禁止されています。

画面モードによっては，グラフィックのページ数が１ページや２ページしかない場合もあります。１ページだけの画面モードのときには，プライオリティ値とページ番号がすべて一致した値，$E４を書き込みます。２ページのモードのときには，GP０とGP１が，GP２とGP３がペアとなり，GP０とGP１のペアがプライオリティの高い側，GP２とGP３のペアがプライオリティの低い側のページ番号設定に使用されます。ページ０を指定するには0100を，ページ１の指定は1110を指定します。つまり，ページ０のプライオリティがページ１よりも高い場合には$E４（11100100）を，逆の場合には$4Ｅ（01001110）を指定することになります。

3 画面間プライオリティ設定

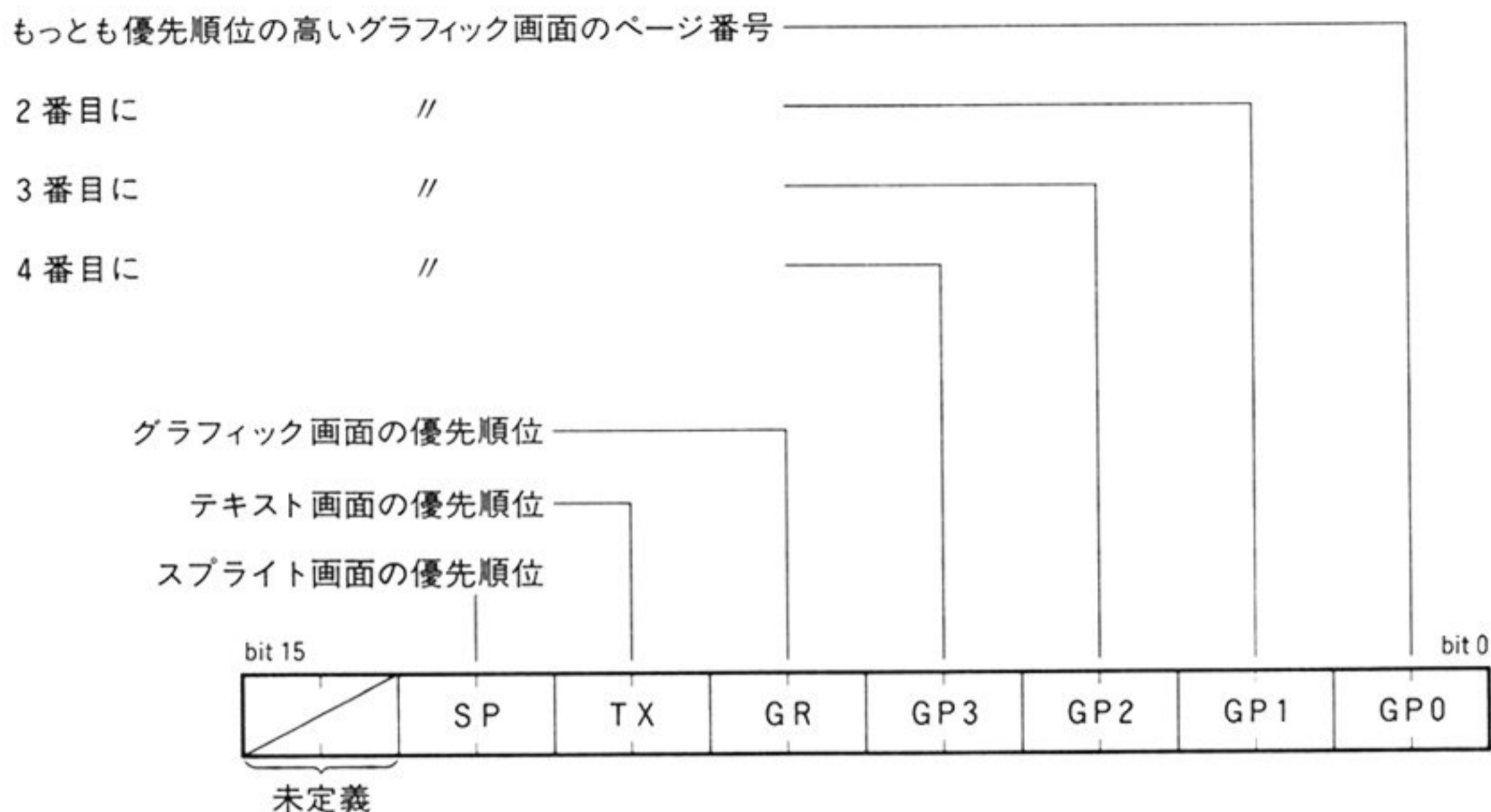
Ｒ１の上位８ビットでは，グラフィック，テキスト，スプライト＋BGの各画面のプライオリティ設定を行います。プライオリティ値は'00'がもっともプライオリティが高く，'01'がその次，'10'がもっとも低いという指定になります。'11'という設定は禁止です。異なる画面に同じプライオリティを設定することも禁止されています。

Ｒ１の最上位の２ビット（ビット14とビット15）は現在使用されていませんので，何を書き込んでも動作には影響しません。

4 ON/OFF設定

ビデオコントローラのＲ２の下位８ビットはグラフィック，テキスト，スプライト＋BGの各画面の表示ON/OFF制御を行います。

グラフィック画面のON/OFFは，実画面が1024×1024ドットのとき（Ｒ０のビット２が１

●図……17 ビデオコントローラ R1($E82500)

〔4ページモード以外でのGP3～GP0の設定〕

プライオリティは2bitで表され

00＞01＞10

の順になる。('11'は設定禁止)

のとき)にはビット4で，実画面が512×512ドットのときにはビット0～3を使って，ページごとのON/OFF制御が行えます。ビットが'1'になっていると表示がON，'0'だとOFFになります。このON/OFF制御用のビットは，各ページに対応するのではなく，プライオリティに対応していることに注意しておいてください。つまり，ビット0でON/OFF制御されるのはグラフィックのページ0ではなく，R1のビット0と1で指定されているページになります。画面モードによってはページ数が4ページ未満のこともあります。このときの設定は次のようになります。

画面が1ページのとき(65536色モード)，ビット0～3はすべて同じ値にします。表示をONにするときは'1111'に，OFFのときは'0000'になります。画面が2ページのとき(256色モード)にはビット0とビット1，ビット2とビット3を同じ値にします。たとえばプライオリテ

ィの高いほうの画面が表示OFFで，低いほうの画面がONなら，設定する値は'1100'になります。

テキスト画面とスプライト＋BG画面は，それぞれR2のビット5と6でON/OFF制御を行います。いずれも'1'のときが表示ON，'0'のときがOFFになります。ビット7は未定義となっていますが，'0'を書き込むようにしてください。

❸·❷ 2 スプライトコントローラの持つON/OFF，プライオリティ制御

スプライトのON/OFFやBG画面との間でのプライオリティの設定は，スプライト1つ1つに対応しているスプライトスクロールレジスタで個別に行い，BG画面のON/OFF制御はBGコントロールレジスタでページごとに行います。それぞれのレジスタの内容を190，191ページの図18と図19に示します。

BG画面間のプライオリティは，つねにBG 0がBG 1よりも高くなっており，変更はできません(画面処理の都合上どうしても入れ替えを行いたいときには，それぞれが使用しているBGデータエリアの番号のほうを入れ替えてしまうことで同じ効果を得ることができます)。

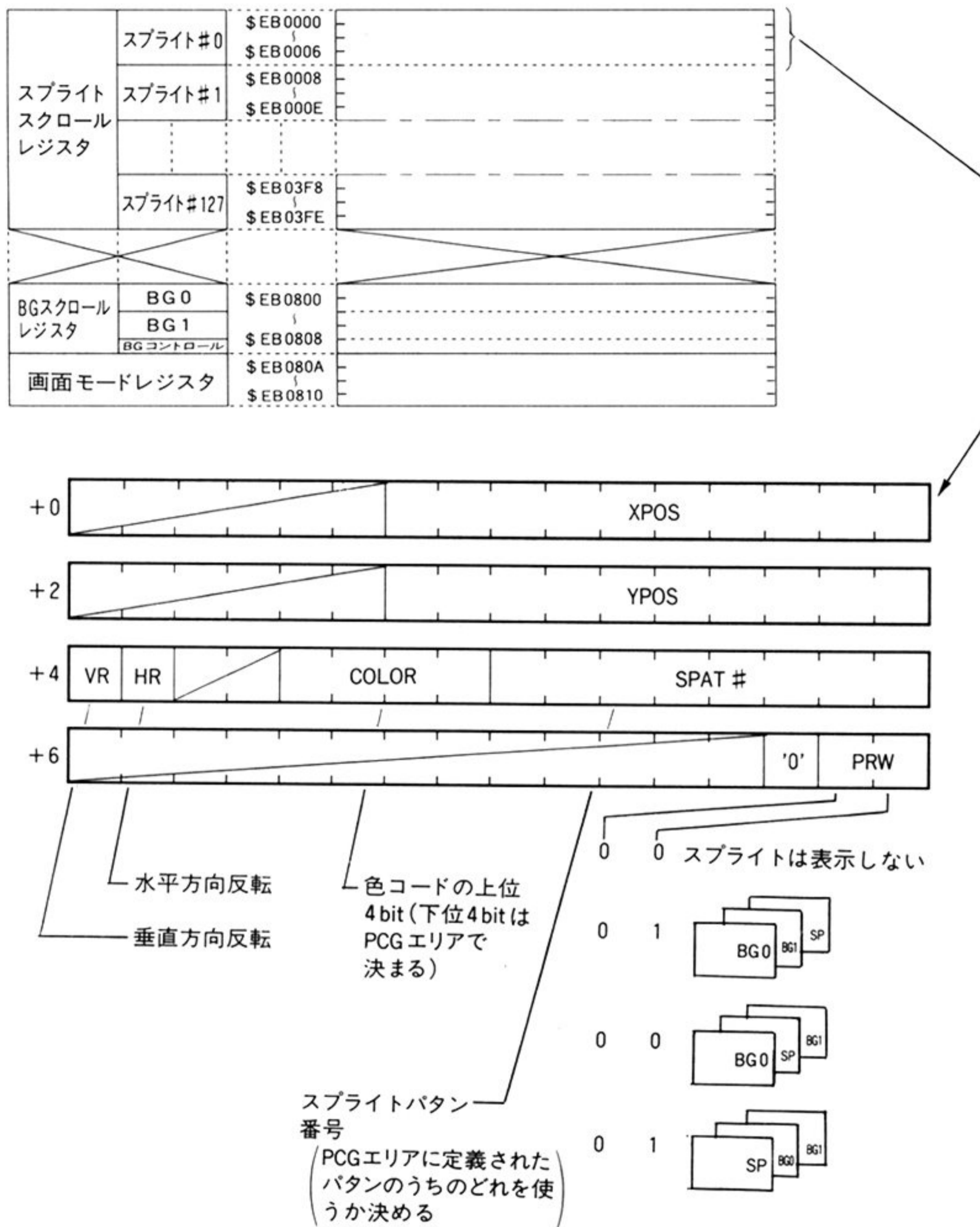
スプライトスクロールレジスタは，各スプライトごとに4ワード（8バイト）分の領域があり，プライオリティ制御とON/OFF制御は4ワード目の下位2ビット（ビット0と，ビット1）に割り当てられています。この2ビットのデータが'00'のときには，該当するスプライトの表示がOFFになります。'01'のときには，スプライトはBG画面の後ろ，'10'のときにはBG 0とBG 1の間，'11'のときにはBG画面の上に表示されます。

BG画面のON/OFFは，BGコントロールレジスタのビット0（BG 0 ON）とビット3（BG 1 ON）によって，各BG画面ごとに独立して行えるようになっています。

●図……18　BG スクロールレジスタ（BG コントロール）（$EB 0808）

＊BG0とBG1で同一のBGデータエリアを使用してもよい。

●図……19　スプライトスクロールレジスタ（$EB 0000〜）

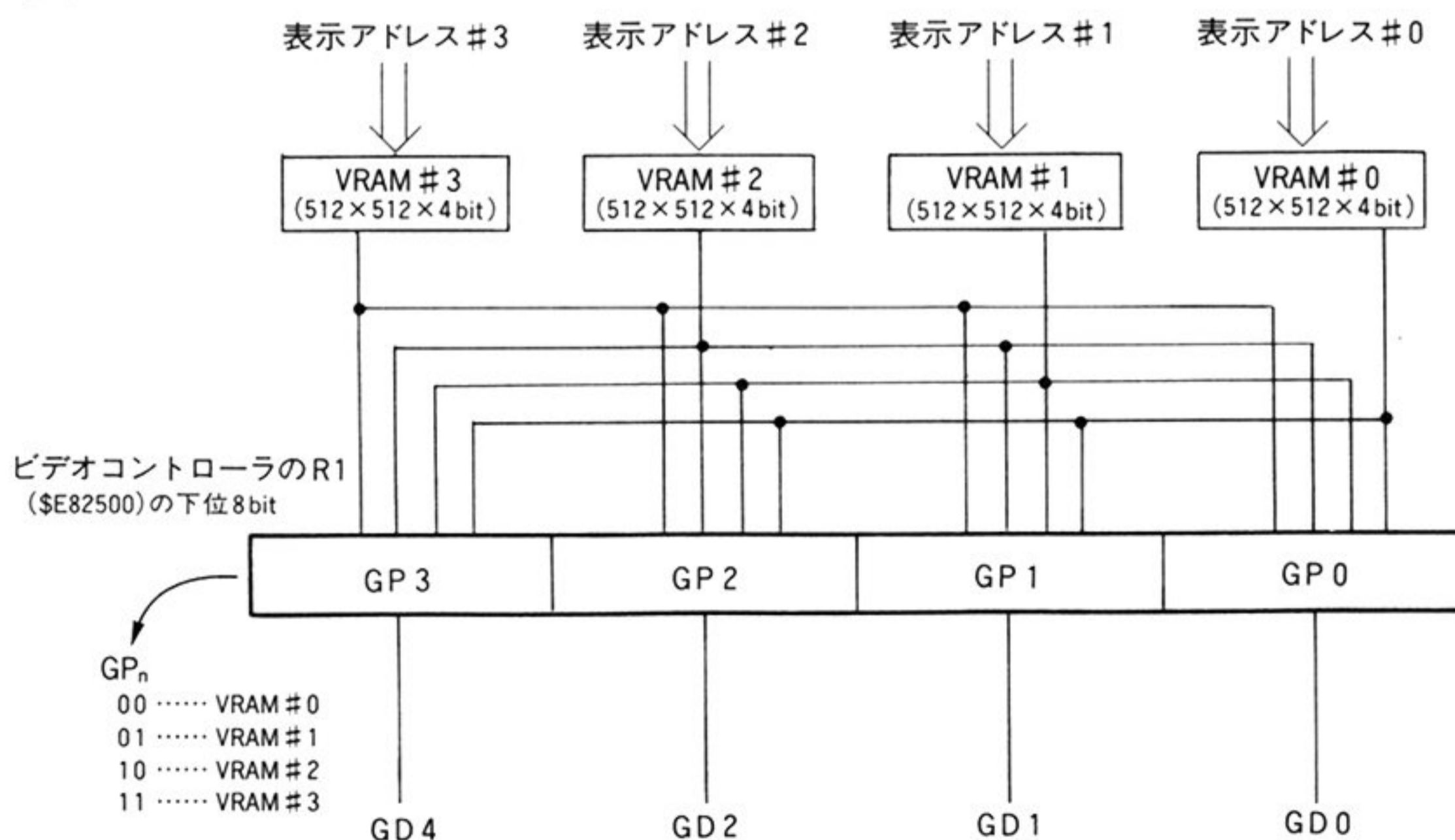
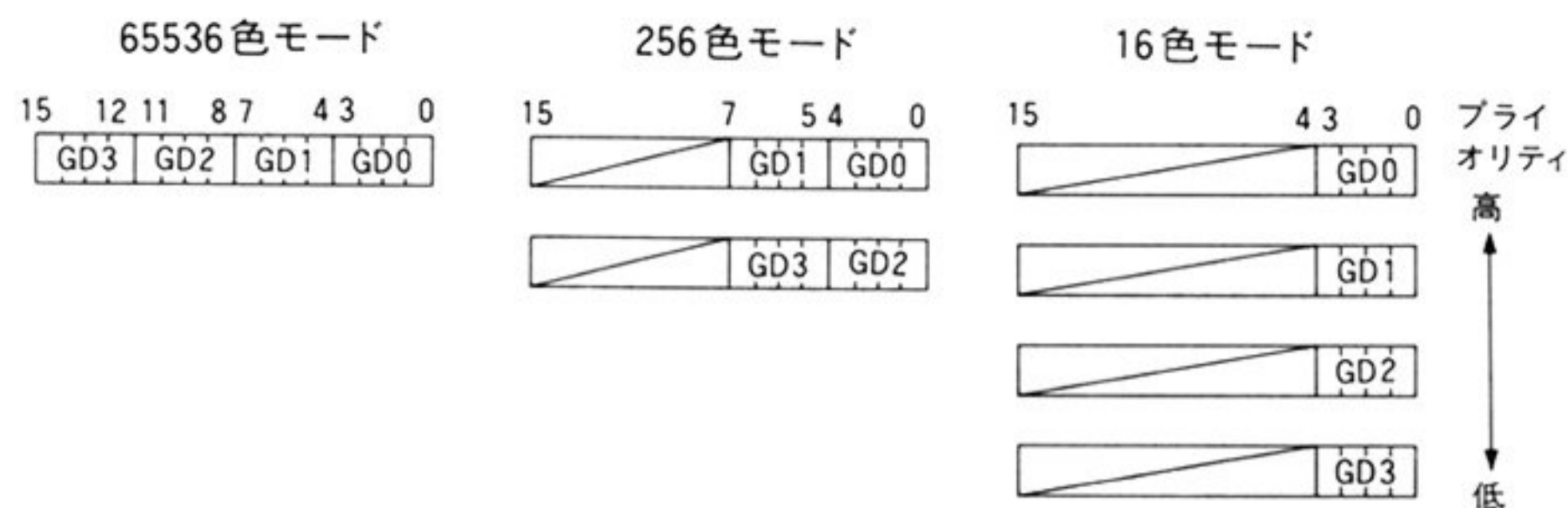
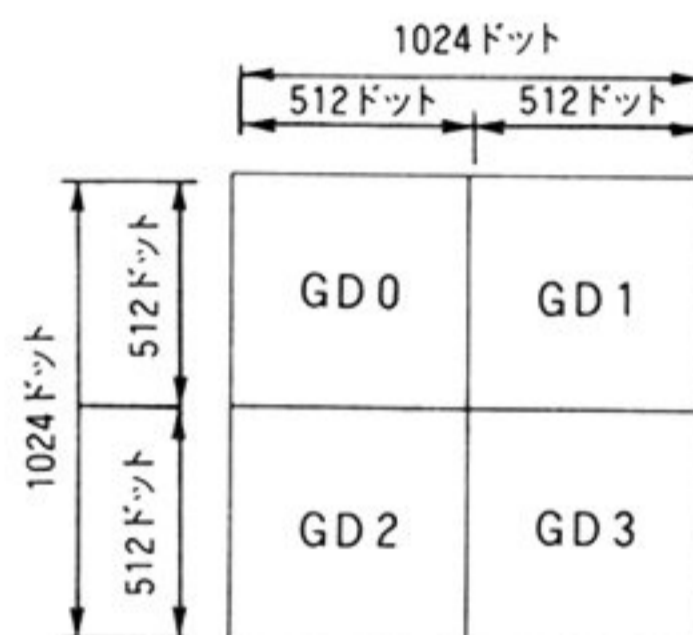
COLUMN

グラフィックページ間プライオリティ制御のからくり

本文中では動作の説明がややこしくなるため，標準設定以外の値を設定した場合，どのような動作になるか触れられなかったので，ここでプライオリティ制御の仕組みとあわせて説明しておくことにしましょう。なお，この内容は筆者が個人的に調べたものなので，将来にわたってこのような仕様である保証はありません。標準設定以外の値を意図的に使うときはこの点に注意しておいてください。

X 68000のグラフィック画面のプライオリティ制御機構のブロック図を図20に示します。

グラフィック画面用のRAMは512×512ドット×4ビット分（128 Kバイト）が1ブロックとなっており，これが4ブロック分集まってグラフィックVRAMを構成しています。図では，このそれぞれにVRAM＃0〜VRAM＃3という番号をつけておきました。CPUからアクセスするときには，256色モードのときにはVRAM＃0と＃2が下位4ビット，VRAM＃1と＃3が上位4ビットとなり，65536色モードのときにはVRAM＃0が最下位の4ビット，VRAM＃3が最上位の4ビットとなるように組み合わされます。

一方，プライオリティ制御回路からの出力も4ビット単位のデータが4つとなっています。これを図ではGD 0〜GD 3で示してあります。このデータの扱われ方は，実画面が512×512ドットのときと，1024×1024ドットのときとで大きく変わります。実画面が512×512ドッ

●図……20　グラフィック画面間プライオリティ制御機構

実画面512×512ドット時

実画面1024×1024ドット時

トのときは，色モードによって次のように変化します。

16色×4ページモード時

GD 0～GD 3がそのまま4つの画面のデータとして扱われます。プライオリティはGD 0がもっとも高く，GD 3がもっとも低くなります。

256色×2ページモード時

GD 1とGD 0，GD 3とGD 2が組み合わされます。GD 0とGD 2が下位の4ビット，GD 1とGD 3が上位の4ビットになります。GD 1とGD 0の組み合わせの画面がGD 3とGD 2の組み合わせの画面よりもプライオリティが高いものとして扱われます。

65536色×1ページモード時

GD 0～GD 3がすべて組み合わされて65536色のデータになります。GD 3が最上位の4ビット，GD 0が最下位の4ビットとなります。

実画面が1024×1024ドットモードのときは，GD 0～GD 3の4つの画面が組み合わされて1024×1024ドットの画面を構成します。組み合わされ方は図の下に示したとおり，GD 0が左上，GD 1が右上，GD 2が左下，GD 3が右下の512×512ドットの領域のデータとなり

ます。

ビデオコントローラの下位8ビットのGP 0～GP 3は，GD 0～GD 3のそれぞれがVRAMの，どのバンクに対応するかを決めているのです。IOCSコールなどで画面を初期化した後は，VRAMのバンク番号とGDが1対1に対応するような値（$E 4：GP 3=11，GP 2=10，GP 1=01，GP 0=00）になっています。本文中では，異なるプライオリティに同一の画面を設定してはいけないと書きましたが，このことを理解して扱うなら，同じ画面を指定してもかまいません。

この値を意図的に書き換えるとおもしろい動作になります。たとえば，256色×2画面モードのときにGP 0～GP 3を$D 8（GP 3=11，GP 2=01，GP 1=10，GP 0=00）にしてみます。これは標準設定からGP 1とGP 2を取り替えたものです。こうすると，プライオリティの高いほうの画面の色コードはRAM＃0を下位4ビット，RAM＃2を上位4ビットとする8ビットデータに，低いほうの画面はRAM＃1を下位4ビット，RAM＃3を上位4ビットとする8ビットデータになります。

実画面が1024×1024ドットのときもGD 0～GD 3の標準設定は$E 4です。これを先ほどと同じように$D 8とすると，GD 1の領域とGD 2の領域がそっくり入れ替わります。また，$00とすると，GD 1～GD 3の領域もすべてGD 0と同じものが表示されることになります。

❸·3 画面スクロール

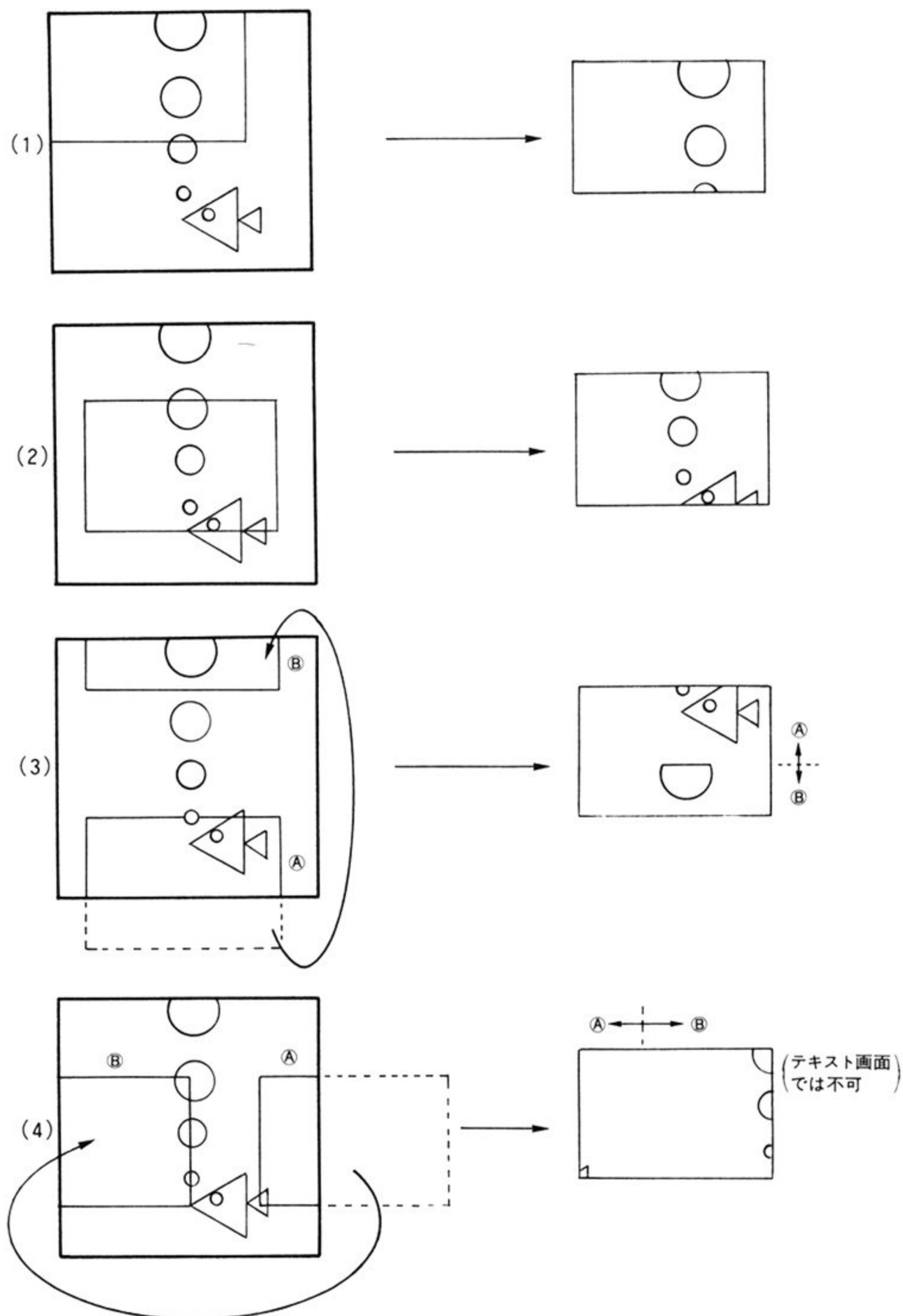
スクロールとは，画面上に表示されているものを全体に上下左右に連続して動かすことです。ソフト的にスクロールを行うときは，実際にVRAMのデータを移動方向にあわせて転送することになりますが，ここで述べるハード的な画面スクロールは，実画面上での表示開始位置（表示画面の左上の座標）を任意に変更することで行います。表示開始位置を実画面で右のほうに動かしていくと，画面上は表示されているものがすべて左に移動しているように見えるため，スクロール動作が実現されるわけです。スクロール機能は，画面の表示を縦や横に移動するスクロール処理の高速化だけでなく，実画面が表示画面よりも大きい場合に実画面上の適当なエリアを表示させるなどの用途にも用いられます。X 68000ではテキスト画面，グラフィック画面の表示開始位置はCRTCで，BG画面はスプライトコントローラのBGスクロールレジスタで行います。

表示開始位置の指定はテキスト画面，グラフィック画面，BG画面それぞれで独立して行え，さらに，グラフィック画面やBG画面が複数ページある画面モードのときには，各ペー

ジごとに指定できるようになっています。各画面とも，表示画面は実画面内いっぱいまで自由に動かすことができます。表示画面の範囲が実画面からはみ出すような指定をした場合の動きに違いがあります。たとえば，実画面の水平ドット数が512ドット，表示画面の水平ドット数が256ドットのときに，表示開始位置のX座標として257以上の値を与えると，表示画面の右端の位置は実画面の外側にはみ出します。

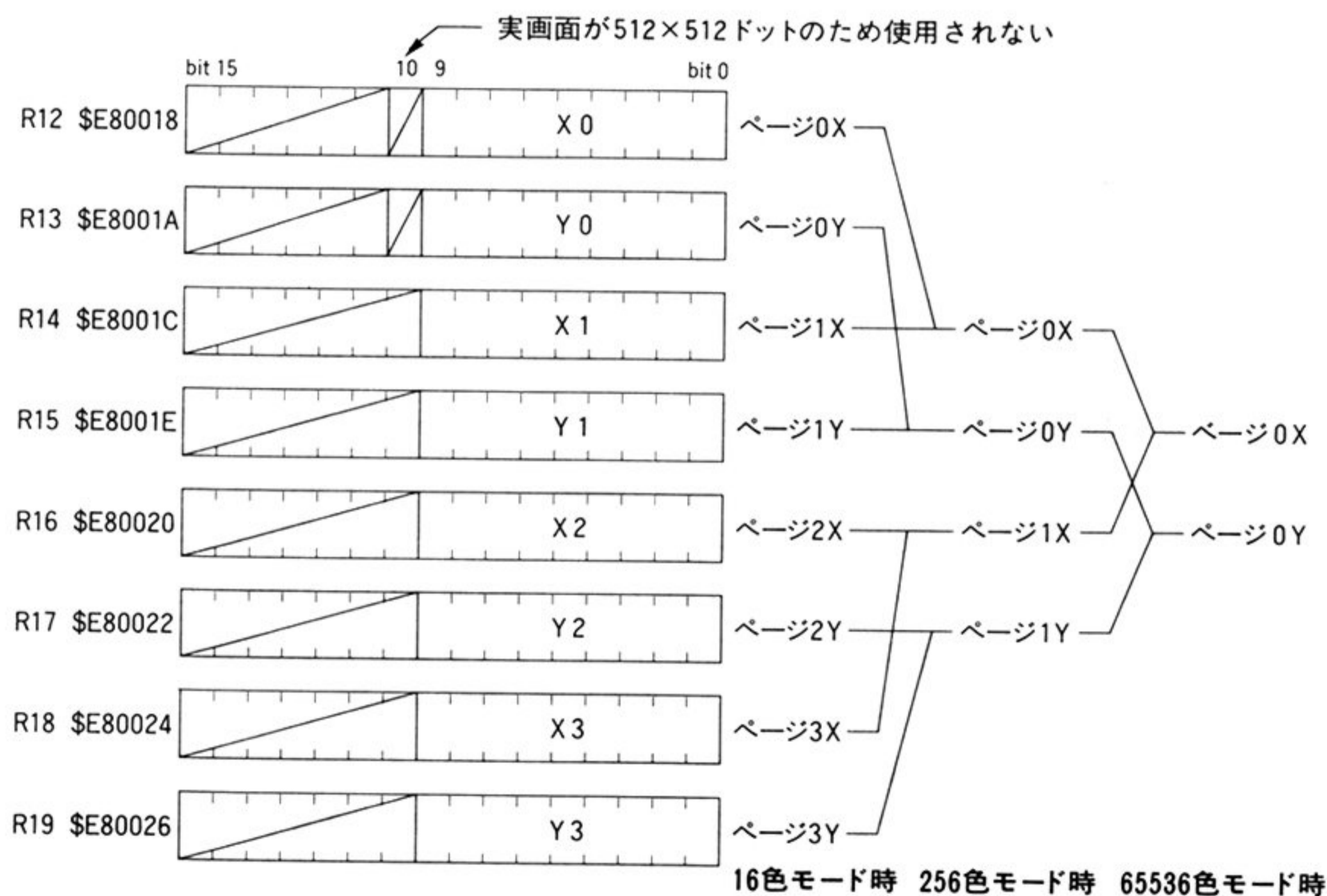
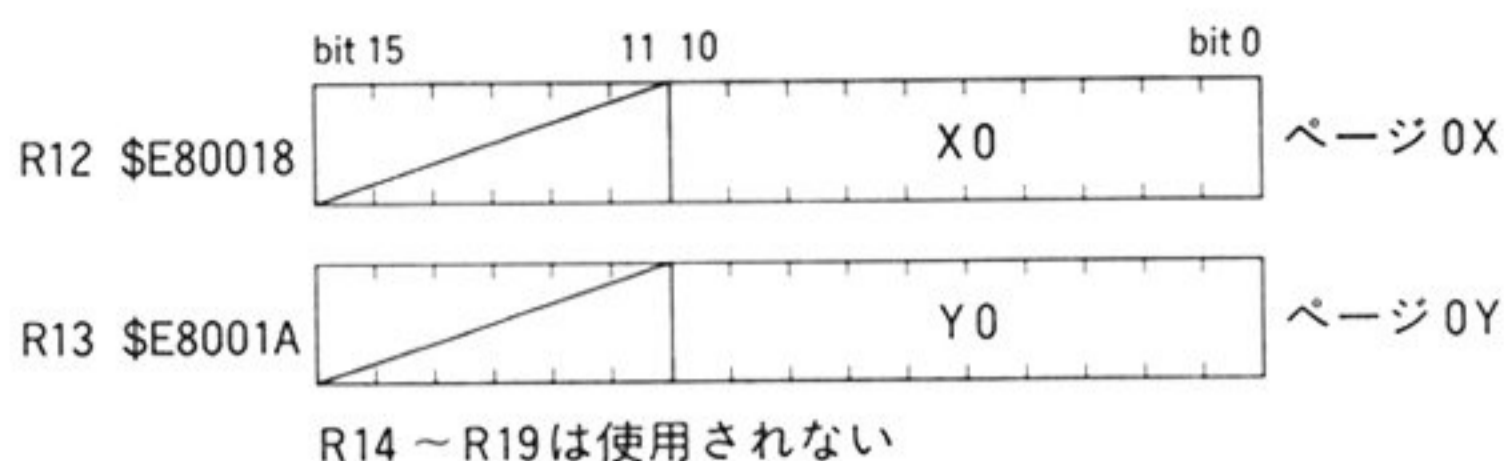
このときの動作を図21に示します。グラフィック画面やBG画面では，上下左右どちらにもはみ出した指定ができます。はみ出した部分には実画面の反対側にあたる部分が表示されます。上にはみ出した部分は下側の部分が，右にはみ出した部分は左端の部分がつながって表示されます。この性質から，グラフィックやBG画面のスクロールは球面スクロールであるといっています。

一方，テキスト画面は上下方向のはみ出しだけが許され，左右方向にははみ出した指定はできません。はみ出した指定をすると，画面の表示がおかしくなります。上端と下端がくっついたように見えるため，テキスト画面のスクロール方式を円筒スクロールと呼んでいます。196ページの図21に画面スクロール動作の例を示します。

●図……21　画面スクロール

❸・❸1 グラフィック画面とテキスト画面のスクロール

グラフィック画面とテキスト画面のスクロールはCRTCによって行われます。CRTCの持つレジスタのうち，グラフィック画面のスクロールはR 12～R 19(グラフィックスクロールレジスタ)，テキスト画面のスクロールはR 10とR 11 (テキストスクロールレジスタ) で行います。グラフィックスクロールレジスタのうち，ページ0用にあたるR 12とR 13は，実画面が1024×1024ドットモードのときに対応するため，それぞれ10ビットが有効ですが，ページ1～3は実画面が512×512ドット以下のときにしか存在しないので，対応するスクロールレジスタは9ビットまでが有効となっています。

実画面サイズが512×512ドットのときのグラフィックスクロールレジスタの設定は少々注意が必要です(図22参照)。グラフィック画面が16色×4ページモードのときにはR 12, R 13がページ0，R 14,15がページ1にというぐあいに1対1に対応しますが，256色×2ページモードのときにはR 12～R 15がページ0用，R 16～R 19がページ1用のスクロールレジスタと

●図……22 CRTコントローラ グラフィックスクロールレジスタ（$E80018～$E80026)

実画面 512×512ドット時

実画面 1024×1024ドット時

なります。ページ0をスクロールするときには，R 14にはR 12と同一の値を，R 15にはR 13と同じ値を設定します。同様にページ1のときはR 16とR 18，R 17とR 19は同一の値を設定するようにします。

65536色×1ページのときにはR 12，R 14，R 16，R 18のすべてにX座標を，R 13，R 15，R 17，R 19のすべてにY座標を設定します。

実画面が1024×1024ドットのときにはR 12とR 13だけが使用され，R 14～R 19は無視されますので，このような配慮は不要です。

C O L U M N

グラフィック画面のスクロールと高速クリア制御のからくり

画面スクロールや高速クリアのページ選択で指定以外の設定を行うとどのようになるかを説明しておきましょう。これもプライオリティ制御機構と同様に筆者が個人的に調べただけなので，機種の追加などで変更されないという保証はないことに気をつけておいてください。

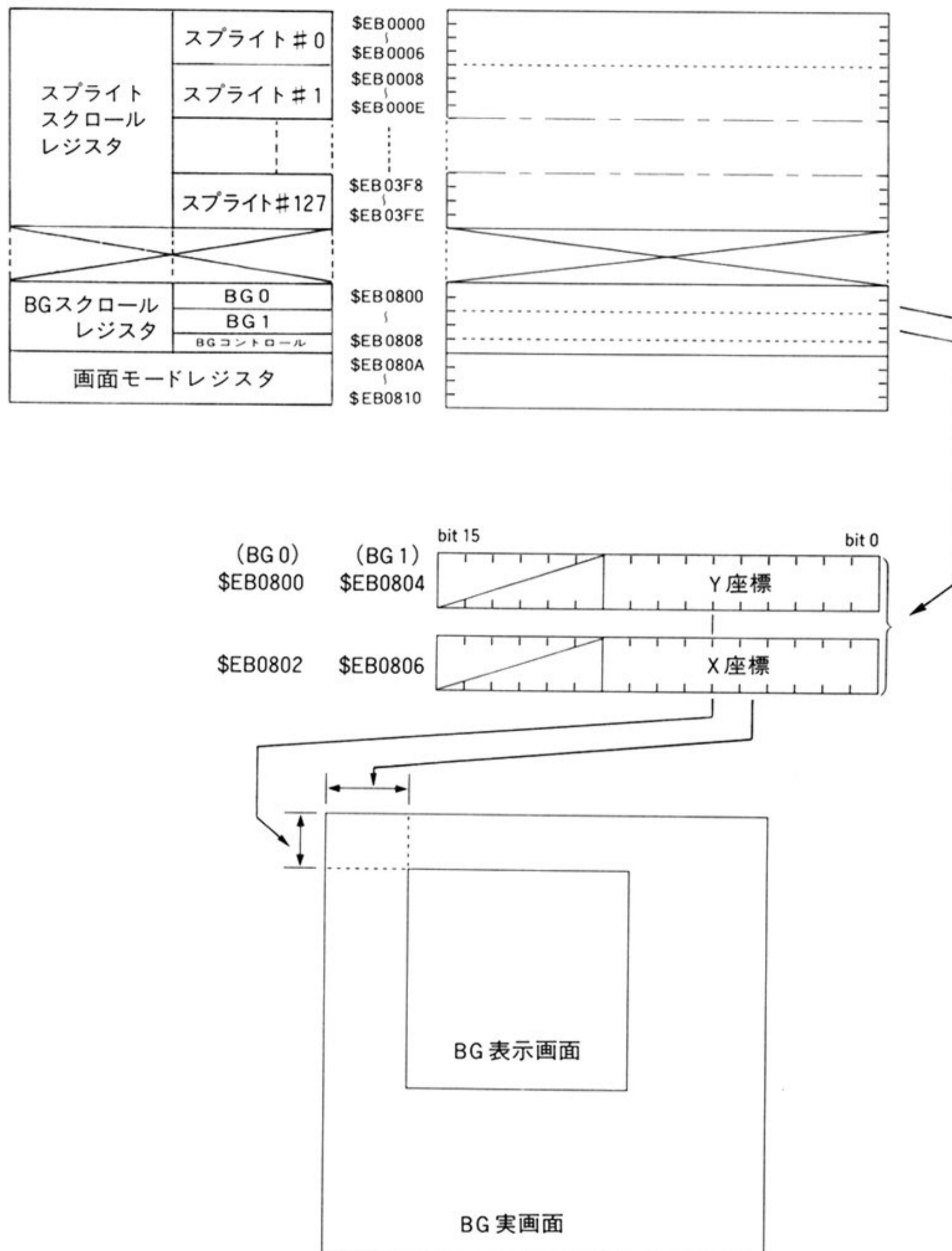
まず，プライオリティ制御のところで示した図20（192ページ）を参照してください。CRTCの中に4組あるグラフィックスクロールレジスタは，それぞれVRAM＃0～VRAM＃3に対応しており，それぞれの表示開始アドレスを変化させるために使用されています。

R 12やR 13を変化させると，VRAM＃0の開始アドレスだけが変化し，R 14，R 15でVRAM＃1のアドレスが変化します。ビデオコントローラのR 1が通常設定になっていると，256色×2ページのときにはVRAM＃0と＃1，VRAM＃2と＃3がペアとなり，65536色×1ページモードのときにはVRAM＃0～＃3がペアとなるため，表示開始アドレスのほうもペアどうしでは同じ値を設定するように指定しているわけです。

高速クリアのプレーン設定も，256色や65536色モードのときには複数のビットがペアとなっており，同じ値を設定するように指定されていますが，これもからくりとしては同じようなもので，R 21のビット0～3がそれぞれVRAM＃0～VRAM＃3に対応しています。

❸·❸ 2 BG画面のスクロール

BG画面のスクロールは，スプライトコントローラの中のBGスクロールレジスタ($EB 0800〜$EB 0807)によって行います。BG画面はBG 0とBG 1の2画面あり，それぞ

●図……23　BGスクロールレジスタ（$EB0800〜$EB0807）

れに対応してスクロールレジスタがあります。各レジスタのビット配置を199ページの図23に示します。

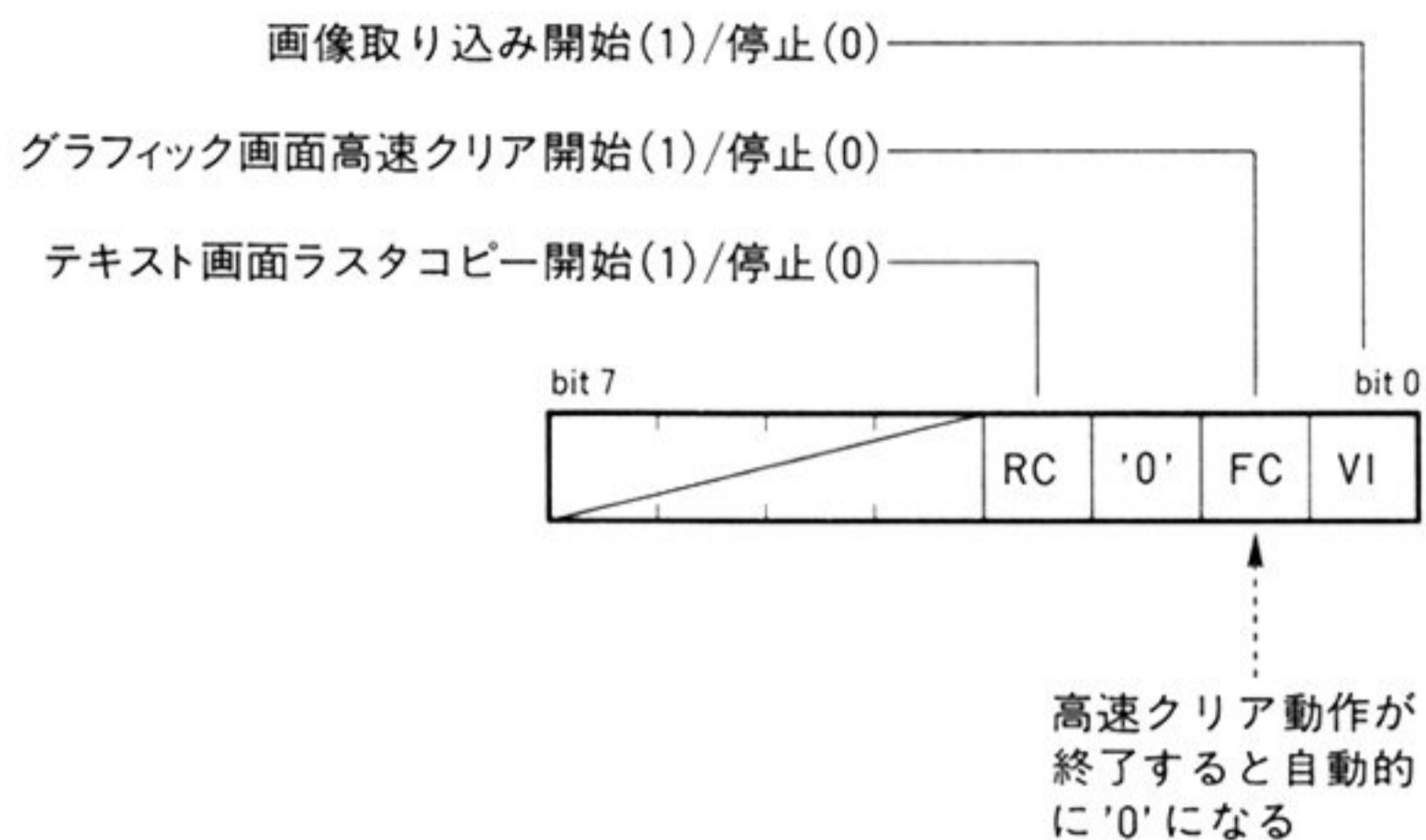
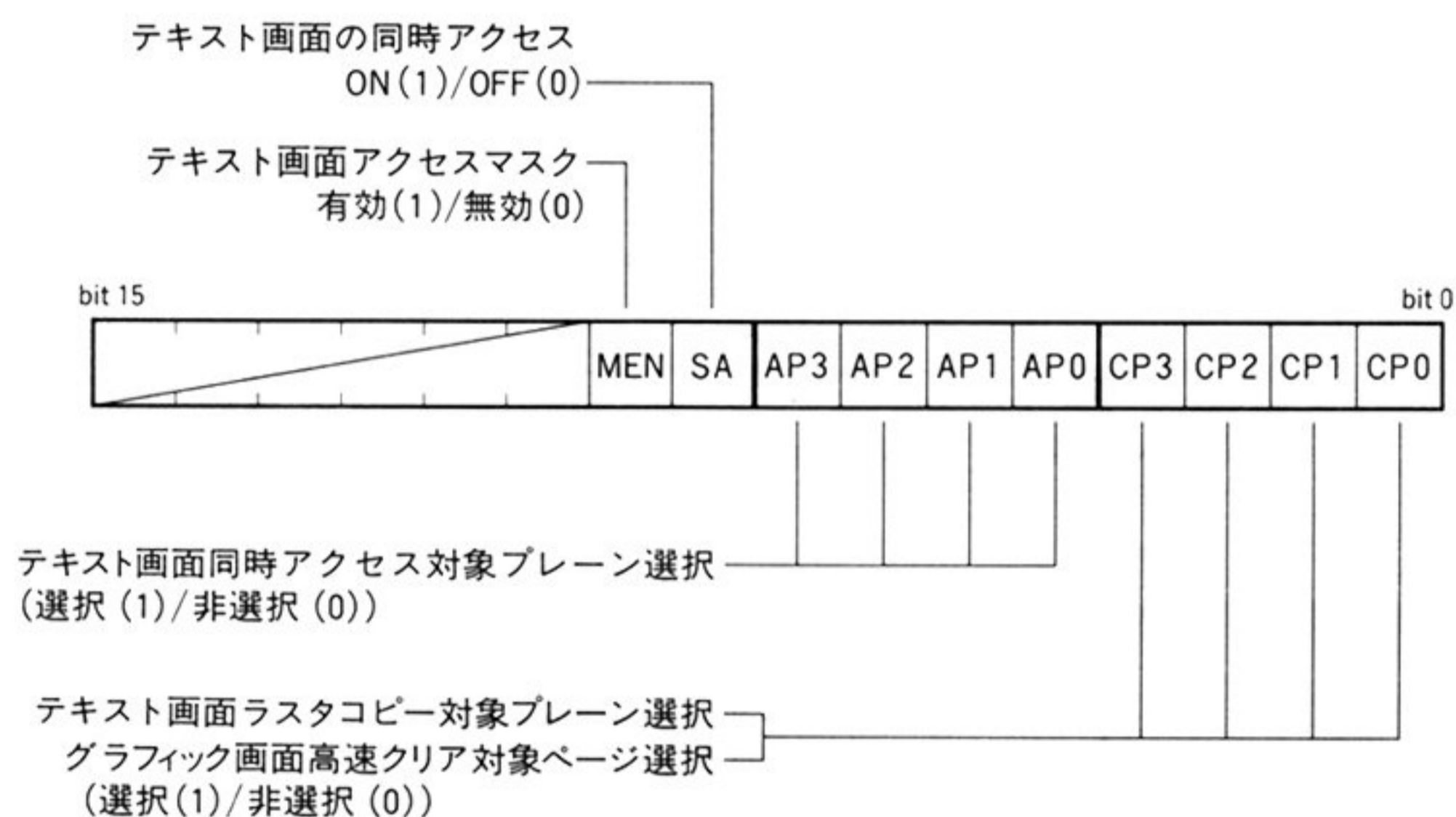
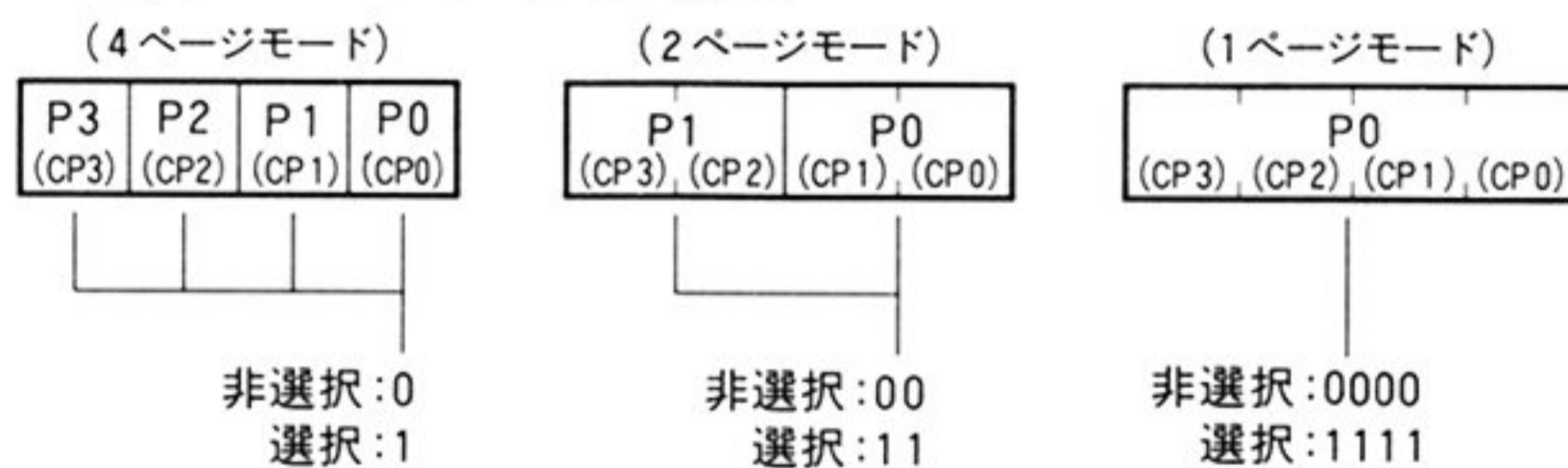
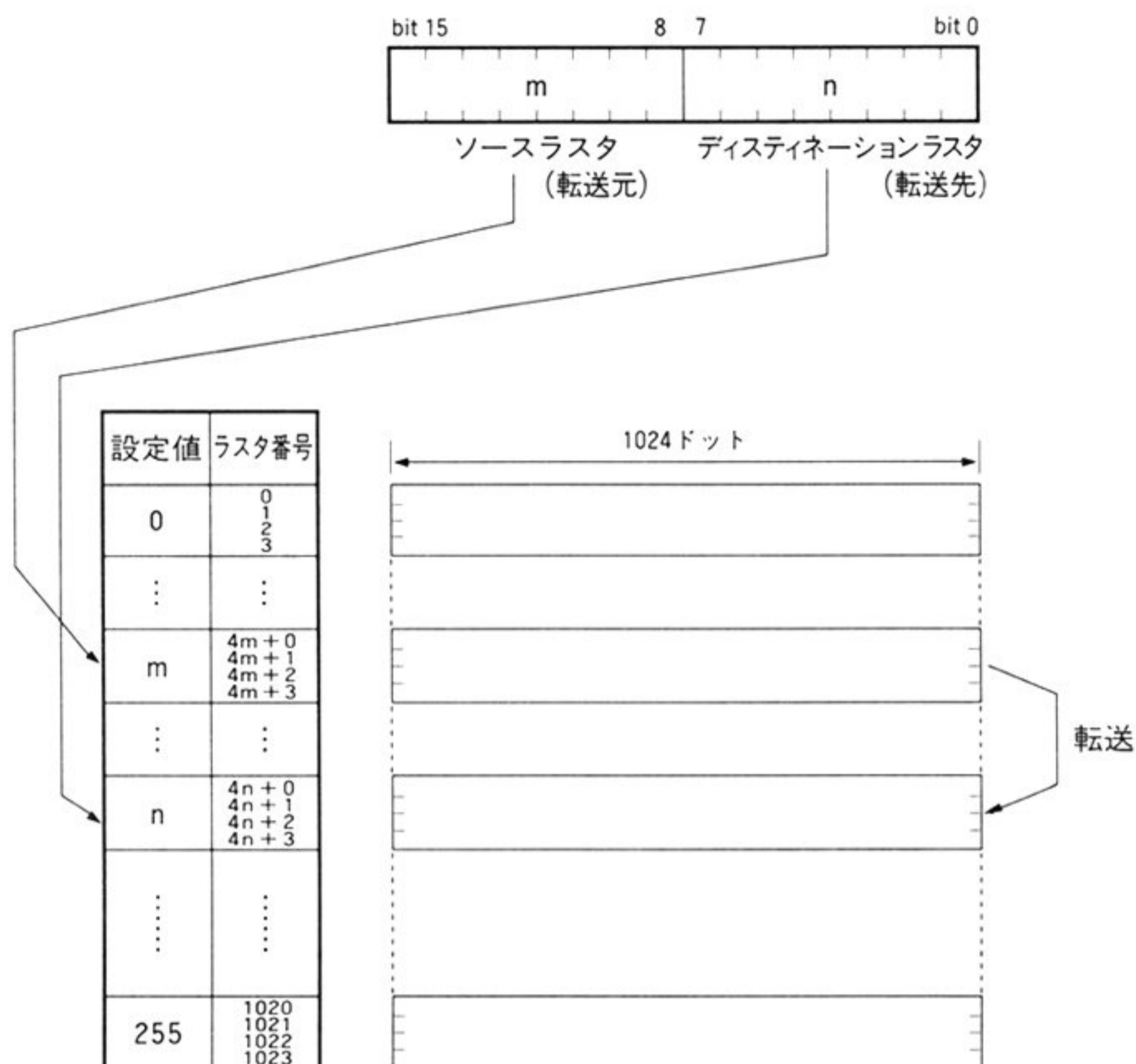
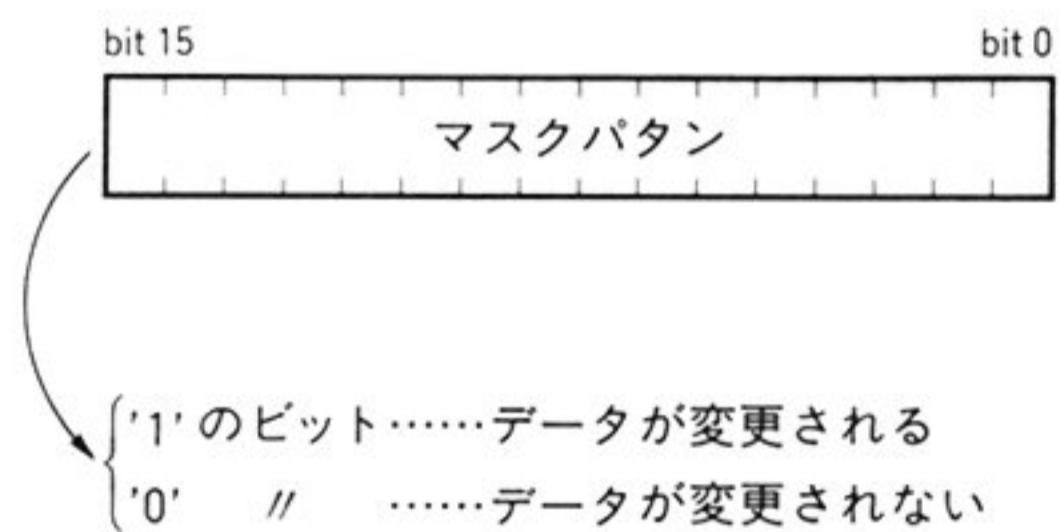
表示画面の水平512ドットモード(BGの実画面1024ドットモード)のときにはBG 0画面のみが表示され，BG 1画面は水平256ドットモード（同実画面512ドットモード）のときにだけ表示されます。つまり，BG 0のスクロールレジスタは10ビットまで有効ですが，BG 1用は9ビットまでが有効ということになります。

❸·4 CRTCの特殊機能

X 68000のCRTCは表示タイミングの発生だけでなく，表示用デュアルポートメモリの特徴を生かした高速画面クリアや画像取り込み，ビットマスクなどの機能も実現しています。ここではCRTCが実現した，これらの特殊機能について説明していくことにしましょう。

CRTCの持つレジスタのうち，特殊機能に関係するものはCRTC動作ポートとR 21～R 23の4つです。それぞれのビット配置を201，202ページの図24，図25，図26，図27に示します。

＊ X 68000の画面表示機構は，一貫してCPUによる画面処理の高速化を主体として考えられています。たとえば，画面構成ではテキスト，グラフィック，スプライト，BGと，目的に応じたさまざまな種類の画面を同時に扱うことができるようにしていました。CRTCの特殊機能は，グラフィック画面やテキスト画面といった，どうしても大量のメモリ操作を必要とする画面の操作の際にCPUの負荷を減らすために設けられた機能です。

●図……24　CRTC 動作ポート（$E80480）

●図……25　CRTC　R21($E8002A)

〔グラフィック画面モードとCP3～CP0の設定〕

●図……26　CRTC　R22（ラスタコピー転送先，転送元指定）（$E8002C）

●図……27　CRTC　R23　テキストアクセスマスク（$E8002E）

❸·❹1 グラフィック画面用の特殊機能

1 グラフィック画面の高速クリア

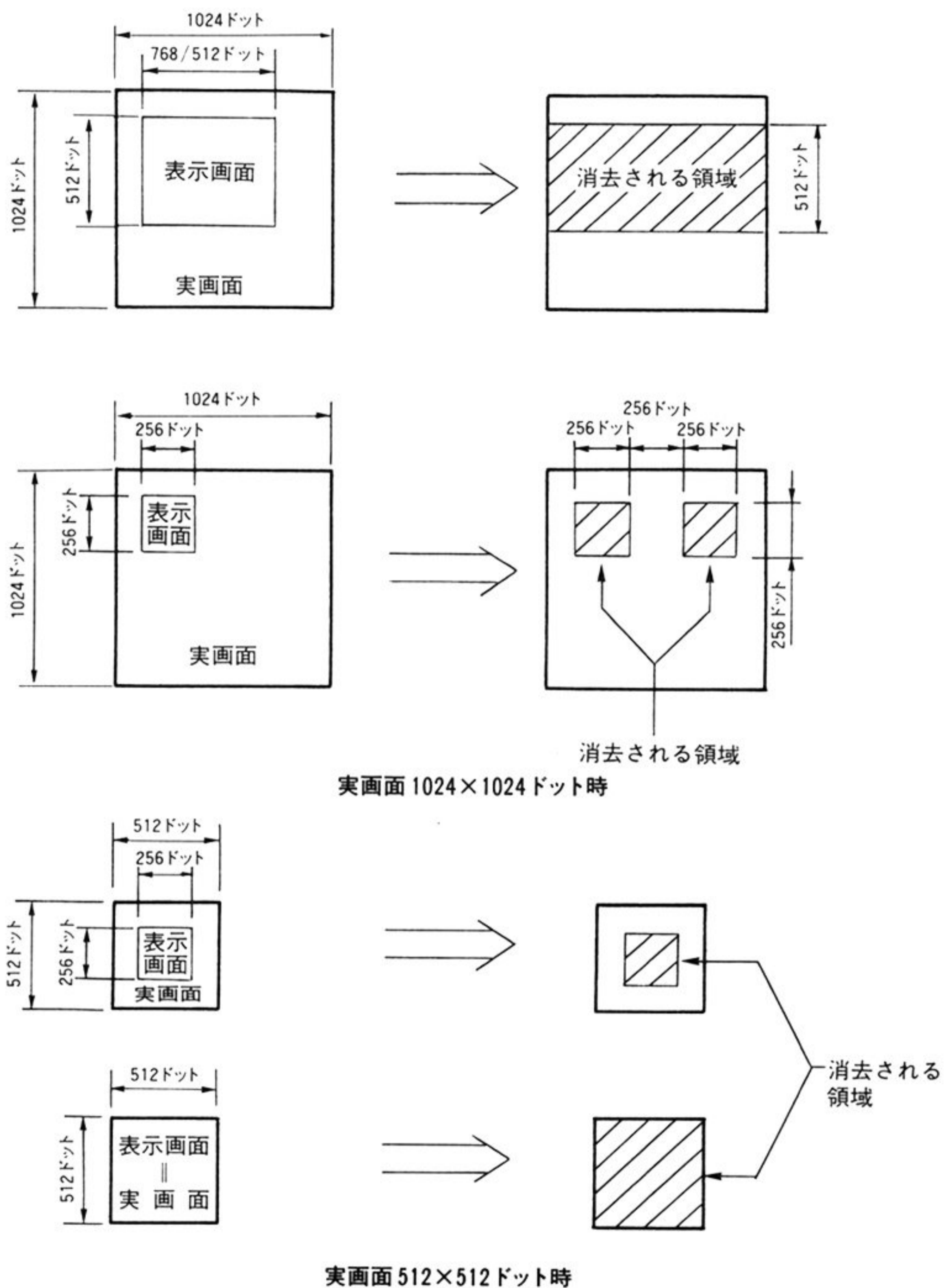
グラフィック画面高速クリアは，グラフィック画面をハード的に高速にクリアする機能です。X 68000は，グラフィック画面用のVRAMとして512Kバイトものメモリを持たせています。しかも，グラフィック画面は1ドットがつねに1ワードという構成になっているため，表示画面が768×512ドットモードのときには表示されているアドレス領域は768Kバイト(768×512×2バイト)あることになります。画面のクリアのたびに，これだけの領域にCPUがアクセスしなくてはならないようでは，速度的にも，CPUの使用効率上もよいことではありません。このため，X 68000ではCRTCが持っている画像取り込み機構の動作を利用して画面の1フレーム分の時間（通常，垂直同期期間1回分，インターレース時は2回分）でグラフィック画面をクリアしてしまう機能を持たせています。この機能を高速クリア機能と呼びます。

高速クリア動作は，グラフィックコントローラのR21の下位4ビットでクリアするページを指定し，CRTC動作ポート（$E80480）のビット1を'1'にすることで，クリア動作の開始を指示します。CRTC動作ポートは，バイト（8ビット）ポートであることに注意してください。高速クリア動作が終了すると，CRTC動作ポートのビット1は自動的に'0'に復帰します。

グラフィック画面の実画面サイズが512×512ドットのときには問題なく，指定したページの実画面全体がクリアされますが，1024×1024ドットのときにはクリアされない領域が残ることに注意が必要です(図28)。表示画面が512×512ドットのときには，縦方向は表示画面の縦方向分(512ドット)，横方向は実画面の幅いっぱいにあたる方形のエリアが消去され，それ以外の部分はそのまま残ります。表示画面サイズが256×256ドットのときには，縦方向はやはり表示画面分（256ドット）ですが，横方向は表示画面の外側左右256ドット分も消去されずに残ってしまいます。

2 画像取り込み

画像取り込みは，オプションのカラーイメージユニットを接続したときに，イメージユニットからX 68000本体に入力される画像データをグラフィックVRAMに転送する機能です。CRTC動作ポート（$E80480）のビット0を'1'にすると，次のV-DISP信号の立ち上がり(フレーム表示期間の開始)時から，このグラフィックVRAMへの転送が始まり，1フレーム分の時間（通常，垂直同期期間1回分，インターレース時は2回分）で1画面全部が取り込まれます。1画面分の取り込みが終了しても，CRTC動作ポートのビット0は'0'に戻らず，取

●図……28　グラフィック高速クリア機能で消去される領域

り込み動作は継続したままになります。取り込み動作を終了させるにはCRTC動作ポートのビット0に'0'を書き込みます。

❸·❹ 2 テキスト画面用の特殊機能

1 アクセスマスク

テキスト画面は，1ワードのデータが画面上で横方向の16ドットに対応する，水平型のビットマップ方式です。このような構造の画面の場合，画面上の1ドットだけを変更したり，水平方向の数ドットだけを変更したりするようなときには，いったんVRAMのデータを読み出し，必要なビットだけを変更したデータをつくってから書き直さなくてはなりません（ウィンドウの端の部分の描画などでは，このようなことが頻繁に発生します)。

X68000では，このような手間を省き，1ワード中の必要なビットだけの書き換えを可能にする，アクセスマスクレジスタ（R23：$E8002E）を用意しています。テキスト画面の書き換えを行う前にアクセスマスクレジスタに，データを変更したいビットを'0'，変更したくないビットを'1'にしたマスクパターンを書き込んでおき，アクセスマスク機能をON（R21：$E8002Aのビット9を'1'にする）にしておくと，以後のテキストVRAMへの書き込みでは，アクセスマスクレジスタで指定したビットだけが書き換わるようになります。

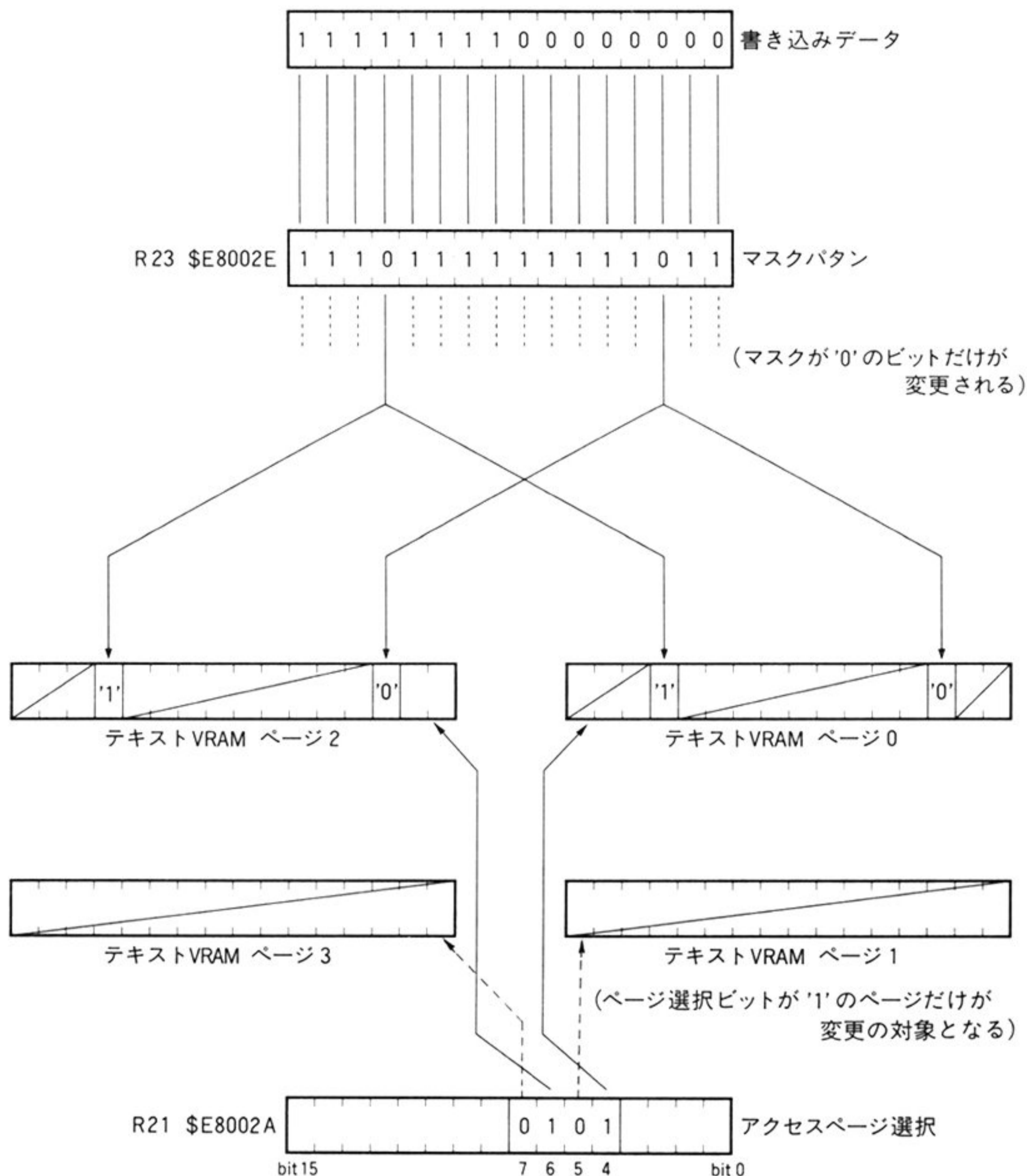
2 同時アクセス

テキスト画面のようなビットマップ画面のもう1つの弱点としてあげられるのが，色指定の面倒さでしょう。X68000のテキスト画面は，4つのプレーンのデータによって色指定を行うようになっています。このため，4つのプレーンすべてを書き換えないと，思いどおりの色に変更できないわけです。

4つのプレーンのデータを変更するのに4回のVRAMアクセスを行うのでは，単純計算でも，表示速度は1/4に低下してしまいます。ただでさえビットマップ画面で処理が重くなりがちな表示速度が，さらに1/4も低下するのはおもしろくありません。また，書き換えに時間がかかっていると，書き換えている間，その部分の色が変化していくのが見えてしまうことになってしまい，見栄えが悪くなります。

このような問題を回避するためにあるのが同時アクセス機能です。同時アクセス機能は，R21のビット4～8で制御されます。ビット8は同時アクセス機能のON/OFFビットで，'1'になっているときだけ，同時アクセス機能が有効になります。

ビット4～7は，同時アクセスするプレーンの選択を行うものです。ビット4～7がそれぞれテキストのT0～T3プレーンに対応しており，同時アクセスを行いたいプレーンに対応するビットを'1'にしておくことで，1回の書き込みで指定したプレーンすべてのデータが書き

●図……29　テキストアクセス制御機構

換わるようになります。

アクセスマスクと同時アクセスの組み合わせによるアクセス制御の例を図29に示しますので参考にしてください。

3 ラスタコピー

テキストVRAMのデータを4ラスタ（水平4ライン）単位で他の任意のラスタ位置に転送

する機能です。もう少しくだけた言い方をすれば，1024×1024ドットあるテキスト画面(実画面)を水平方向に256等分してできる1024×4ドットの横長の長方形エリアを，他の長方形の領域にまるごとコピーする動作です。転送はラスタコピー動作が指示された次の水平同期期間中に行われます。テキスト画面には，グラフィック画面の高速クリアのような機能がありませんが，同時アクセスやラスタコピー動作を利用すれば，グラフィック画面と同等以上の速度でクリアすることができます。

ラスタコピーはR22で転送元と転送先，R21の下位4ビットでラスタコピー動作をさせたいテキスト画面のプレーンの選択を行った後，CRTC動作ポートのビット3を'1'にすることで動作が開始されます。

転送元，転送先はそれぞれCRTCのR22の上位8ビット，下位8ビットで指定します。設定する値は，ラスタ番号ではなく，画面を4ラスタごとに切った横長の領域の番号です。転送されるラスタ番号は，(設定値×4）ラスタから（設定値×4＋3）ラスタまでの4ラスタ分となります。

R21の下位4ビットは，ラスタコピー動作の対象となるプレーン番号の設定です。T0〜T3の各プレーンがビット0〜3に対応しており，'1'を設定したプレーンだけラスタコピー動作が行われます。

❸·5 ビデオコントローラの特殊表示機能

前にも述べたとおり，ビデオコントローラはX68000内部でつくられたテキスト画面，グラフィック画面，スプライト＋BGの各画面と，外部ビデオ信号をもとに，各画面のON/OFFや半透明，特殊プライオリティなどの制御を行い，実際にCRTディスプレイに表示される信号の作成を行っています。画面のON/OFFやプライオリティ制御機能についてはすでに述べました。ここでは残っていた，半透明機能と特殊プライオリティ機能について説明しておきましょう。

❸·❺1 半透明

半透明機能は，グラフィック画面のうちもっともプライオリティの高いページ（仮りにベースページと呼ぶことにします)と，他の画面の色データを50パーセントずつの割合で加算していく機能です*1。加算は，ディスプレイの原色であるRGBそれぞれで独立して行われますので，ちょうど半透明処理を行う画面の色を平均した色になります。2つの画面で半透明動作を

させているとき，片側が単色だと，ちょうど色付きのセロファン紙を通して見たような感じになります。半透明処理を行う領域の指定は，ベースページのVRAMデータの最下位ビットを'1'にして行います。最下位ビットが'0'の領域では半透明処理は行われず，通常表示になります。このとき，ベースページの表示上は，最下位ビットが強制的に'0'にされた状態になります。このため，ベースページで実際に使用できる色数は，半透明動作を行わないときの半分になります。

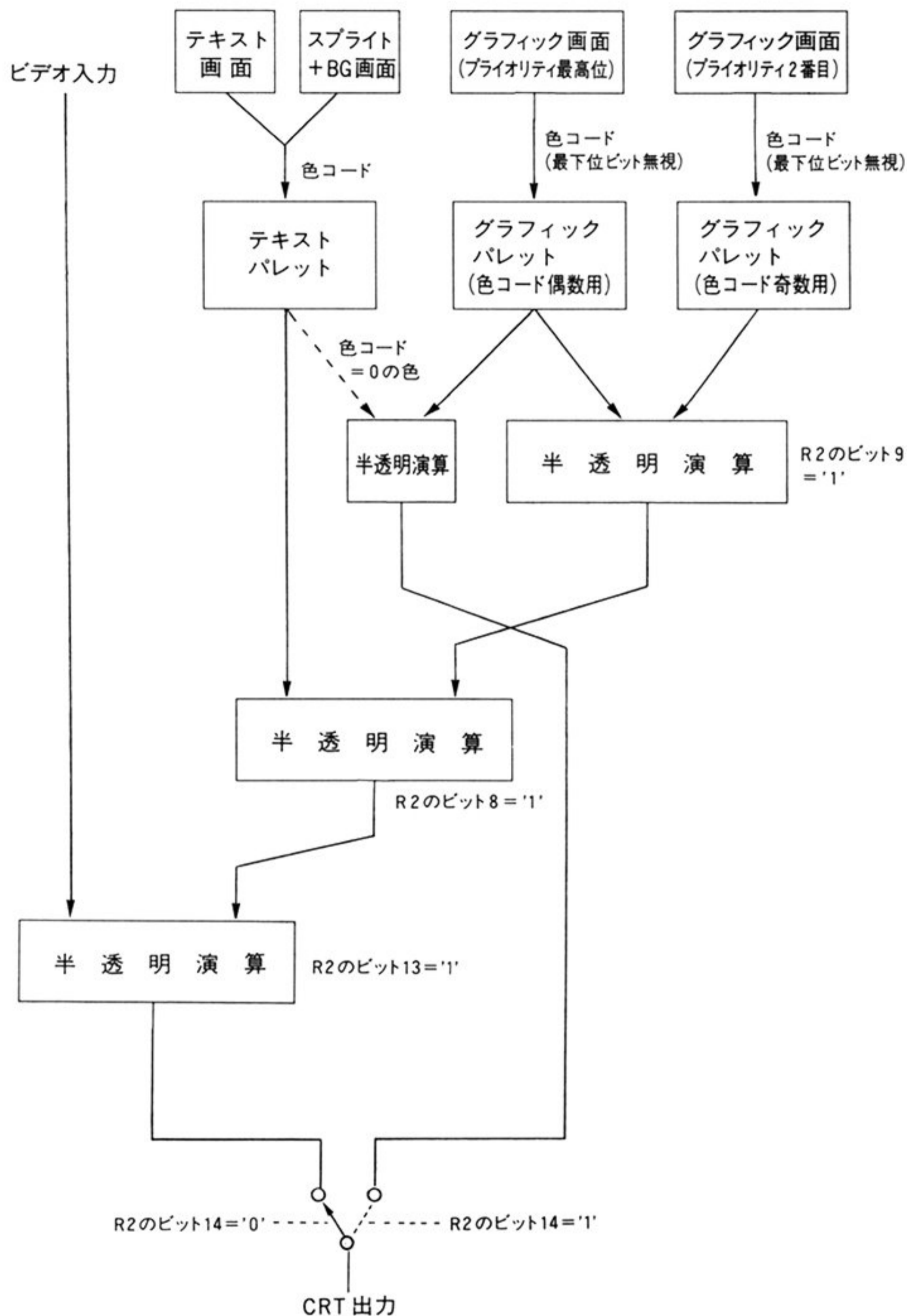
＊1　実際には50パーセントずつにしてから足すのは面倒なので，いったん両方をRGBごとに加算した後で1/2にする（1ビットシフトする）という計算をしています。計算された和の最下位ビットは2で割った場合の余りとなりますが，これは切り捨てられます。輝度ビットはベースページ側は無視され，相手側の輝度ビットがそのまま用いられます。ベースページ側の輝度ビットの値がつねに'1'であると考えると，RGBの計算と同様になります。

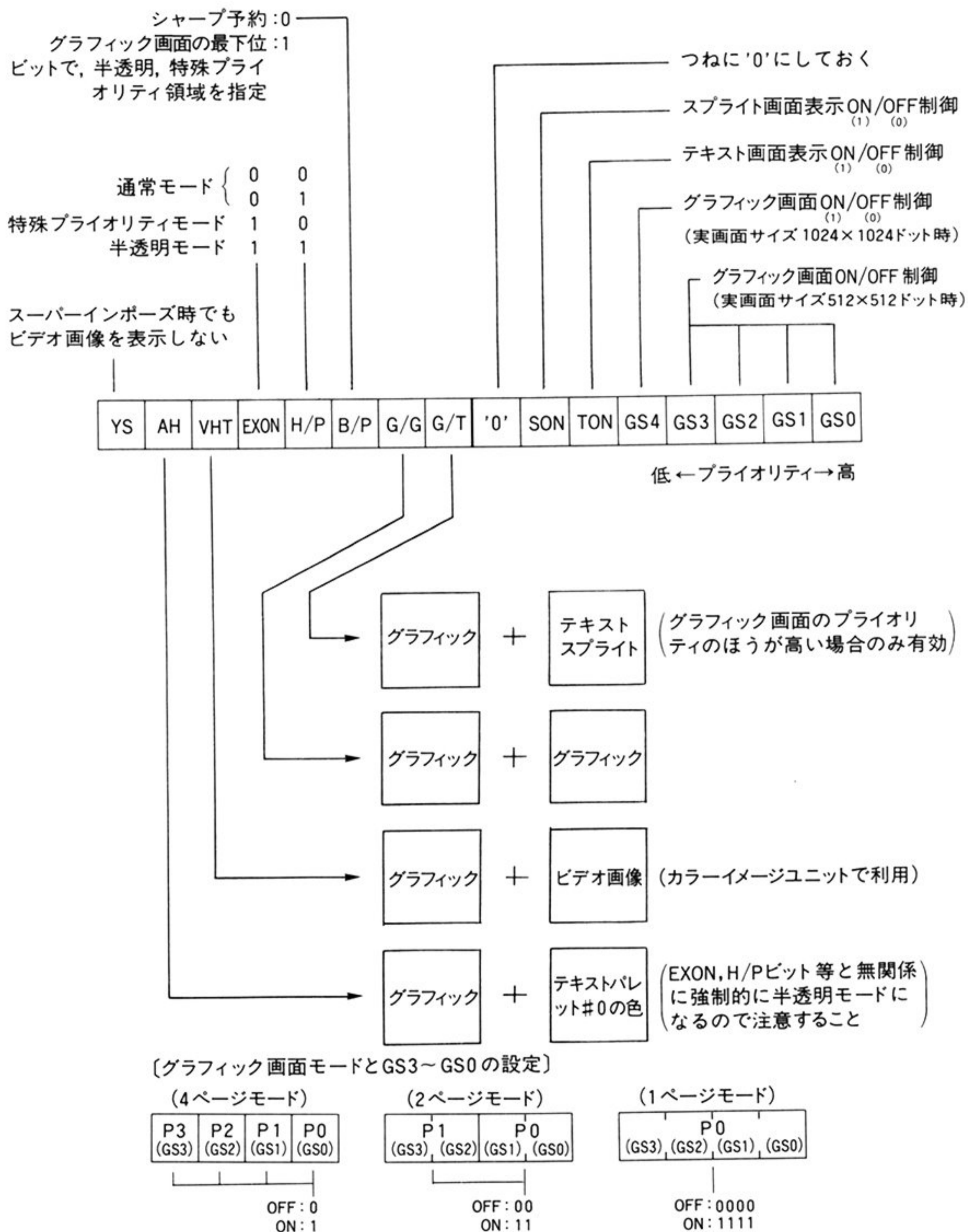
X 68000の半透明機構をデータの流れに注目してまとめると，209ページの図30のようになります。半透明の相手となりうるのは，テキスト(スプライト＋BG画面)＊2，グラフィック画面の中で2番目にプライオリティの高いページ(セカンドページと呼ぶことにします)，テレビ/ビデオ画面，テキストパレット0番の色の計4種類です。このうち，テレビ/ビデオ画面は，オプションで売られているカラーイメージユニットを使用したときに利用されます。

＊2　テキスト画面とスプライト＋BG画面は独立した画面ですが，半透明処理上は連動させられています。テキスト画面を半透明処理の相手にすると，スプライト＋BGも自動的に半透明の対象となります。さらにテキスト画面やスプライト＋BG画面が半透明処理されるのは，ベースページのほうがプライオリティの高い場合だけで，ベースページのほうがプライオリティが低いときには，通常どおりグラフィック画面上に重ねられて表示されます。たとえば，プライオリティの順序がグラフィック>テキスト>スプライト＋BG画面となっていれば，スプライト＋BG画面の上にテキスト画面が重なったものとグラフィック画面の間で半透明処理が行われますし，テキスト>グラフィック>スプライト＋BGの順になっていれば，グラフィックとスプライト＋BG画面が半透明処理されたうえにテキスト画面が重ねられて表示されます。

これらの画面を複数半透明処理対象とすることもできます。ベースページとの間で半透明処理を行わせることのできる組み合わせは，次の7通りがあります。

1)テキストパレット0の色
2)テキスト（スプライト＋BG）画面
3)セカンドページ
4)テキスト（スプライト＋BG）画面　＋　セカンドページ
5)テキスト（スプライト＋BG）画面　＋テレビ/ビデオ画面
6)　セカンドページ　＋テレビ/ビデオ画面
7)テキスト（スプライト＋BG）画面　＋　セカンドページ　＋テレビ/ビデオ画面

●図……30　半透明機構

半透明機能の制御はビデオコントローラのR2で行います。R2のビット構成を図31に示します。

●図……31　ビデオコントローラ R2（$E82600）

半透明機能を使うときにはR2のビット10は必ず'1'にします。このビットが'1'のとき，領域指定をベースページの最下位ビットで指定するということになっています。現在X68000では，領域指定にはこの方法しかサポートされていないので，半透明機能を使うときには'1'以外は選択できません。このビットが'0'のときの動作は未定義となっています。

画面の組み合わせの選択方法は，先ほどの7種類の組み合わせのうちの1)と，それ以外の場合とに分類されます。

ビット14が'1'になっていると，他のビットとは関係なく，無条件に1)が選択されます。

2)～7)の組み合わせの選択時はビット14を'0'に設定します。この場合，さらにビット11と12の両方を'1'にしてビデオコントローラに半透明動作モードであることを教えなくてはなりません。なお，半透明動作が指示されると，半透明対象の画面の有無にかかわらず，自動的にベースページのデータの最下位ビットは'0'であるものとして扱われるようになります。

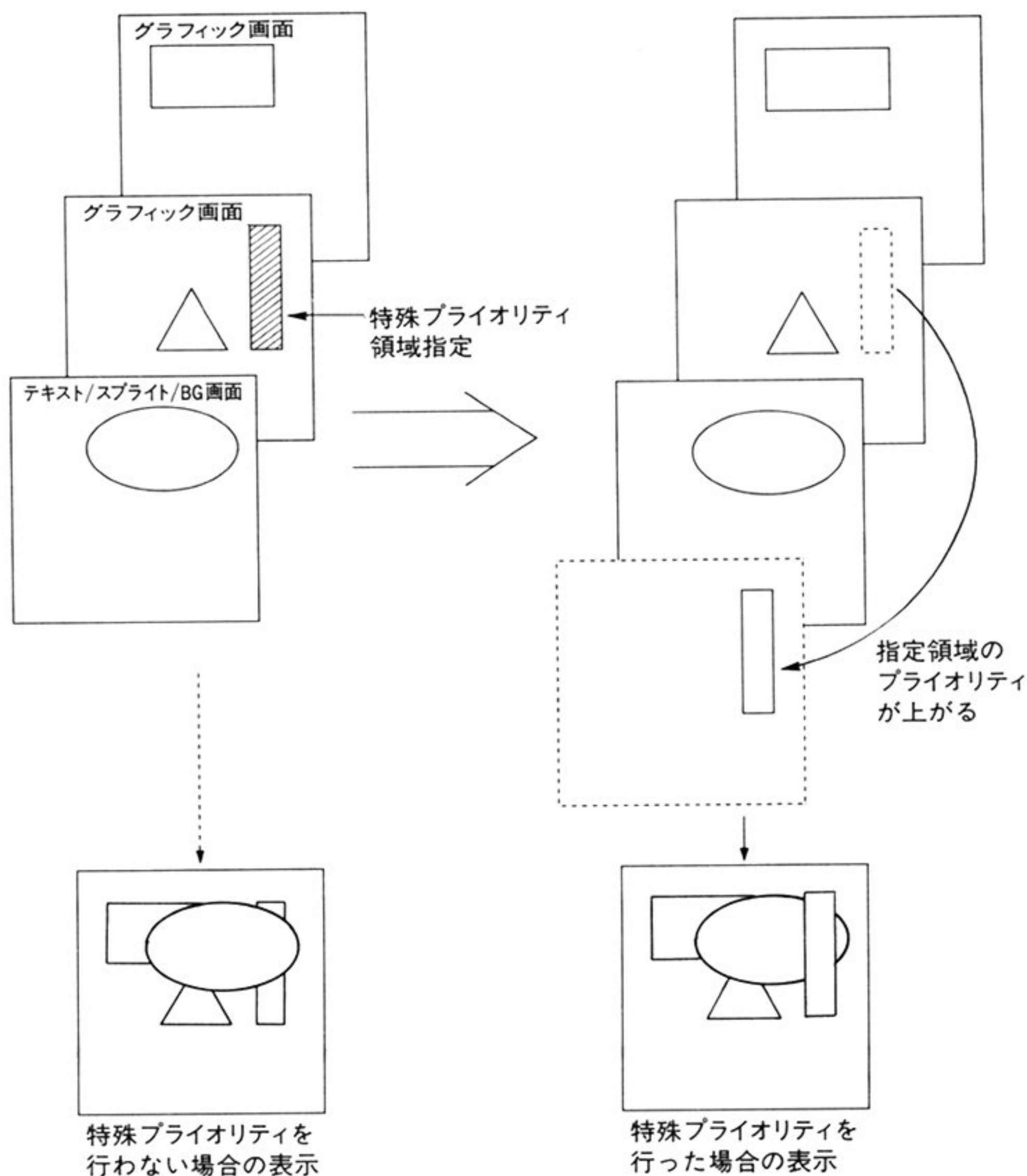
2)～7)の組み合わせからの選択は，ビット8，9，13で行います。それぞれのビットがテキスト画面，セカンドページ，テレビ/ビデオ画面の半透明ON/OFF制御になります。たとえば，4)の組み合わせ，すなわちテキスト（スプライト＋BG）画面とセカンドページの両方との半透明処理を行うときは，ビット8，9，13はそれぞれ'1'，'1'，'0'となります。

ビットパターンからみると，ベースページとテレビ/ビデオ画面だけの半透明もできそうですが，ビデオコントローラ側の制約により，テレビ/ビデオ画面を半透明の対象とするときはテキスト（スプライト＋BG）画面か，セカンドページのいずれかが半透明対象となっていなくてはならなくなっています。つまり，1)以外のパターンではビット8，9のいずれかが'1'になっていないと半透明動作にならないわけです。

❸・❺2 特殊プライオリティ

特殊プライオリティというのは，グラフィック画面のプライオリティがテキスト画面やスプライト＋BG画面よりも低い場合に，グラフィック画面のうちもっともプライオリティの高いページ（半透明機能のときと同じようにベースページと呼ぶことにします）のプライオリティをテキストやスプライト＋BGよりも高くする機能です（212ページの図32参照）。特殊プライオリティ機能と，先ほど説明した半透明機能は選択になっており，両方の機能を同時に使うことはできません。

特殊プライオリティも半透明と同じように特殊プライオリティにする領域を，ベースページのVRAMのデータの最下位ビットで指定します。最下位ビットが'1'になっているドットだけが特殊プライオリティ動作の扱いを受け，テキストやスプライト＋BG画面よりもプライオリティが高くなり，最下位ビットが'0'の部分は通常のプライオリティどおりに表示されます。

●図……32　特殊プライオリティ動作

グラフィック画面のプライオリティ自体がテキスト画面やスプライト＋BG画面よりも高い場合には，当然のことながら特殊プライオリティにはなりません。たとえば，プライオリティの順序がスプライト＋BG＞グラフィック＞テキストならば，特殊プライオリティ領域ではベースページ＞スプライト＋BG＞グラフィック（ベースページ以外）＞テキスト，そのほかの領域ではスプライト＋BG＞グラフィック（ベースページを含む）＞テキストの順になります。

特殊プライオリティ動作の制御はビデオコントローラのＲ２で行います。特殊プライオリティ動作を行わせるには，Ｒ２のビット 14，12，11，10 を '0'，'1'，'0'，'1' に設定します。ビット 10 は半透明のときと同じように，領域指定をベースページの最下位ビットで行うことを示すものですが，現在 X 68000 では，これ以外の方法による領域指定の方法はサポートされて

ので，このビットは半透明機能や特殊プライオリティ機能を使うときには必ず'1'に設定します。

ビット 14 は半透明機能のほうで説明しましたが，このビットが'1'になっていると，強制的に半透明機能（テキストパレット 0 の色との半透明処理）が選択されてしまうため，特殊プライオリティ動作をさせたいときには'0'に設定しておく必要があります。

❸·6 カラーパレット

カラーパレット(以後，たんにパレットと略します)は，VRAM などから出力されるデータ(以後，色コードと呼ぶことにします）と，実際に D/A 変換されて CRT に送り出されるデータ（色データと呼ぶことにします）とを対応させるものです。ブロック（図 12）からもわかるように，X 68000 の出力段は RGB のそれぞれが 5 ビットと輝度 1 ビットの計 16 ビット，65536 色の表示が可能ですから，パレットは色コードがどの値のときに 65536 色中のどの色を出力するかを決定するものとなっています。

X 68000 には 2 組のパレットがあり，片方はグラフィック画面専用，他方はテキストとスプライト＋BG 画面で共用されています。以下，簡略化のために，前者をグラフィックパレット，後者をテキストパレットと呼ぶことにします。

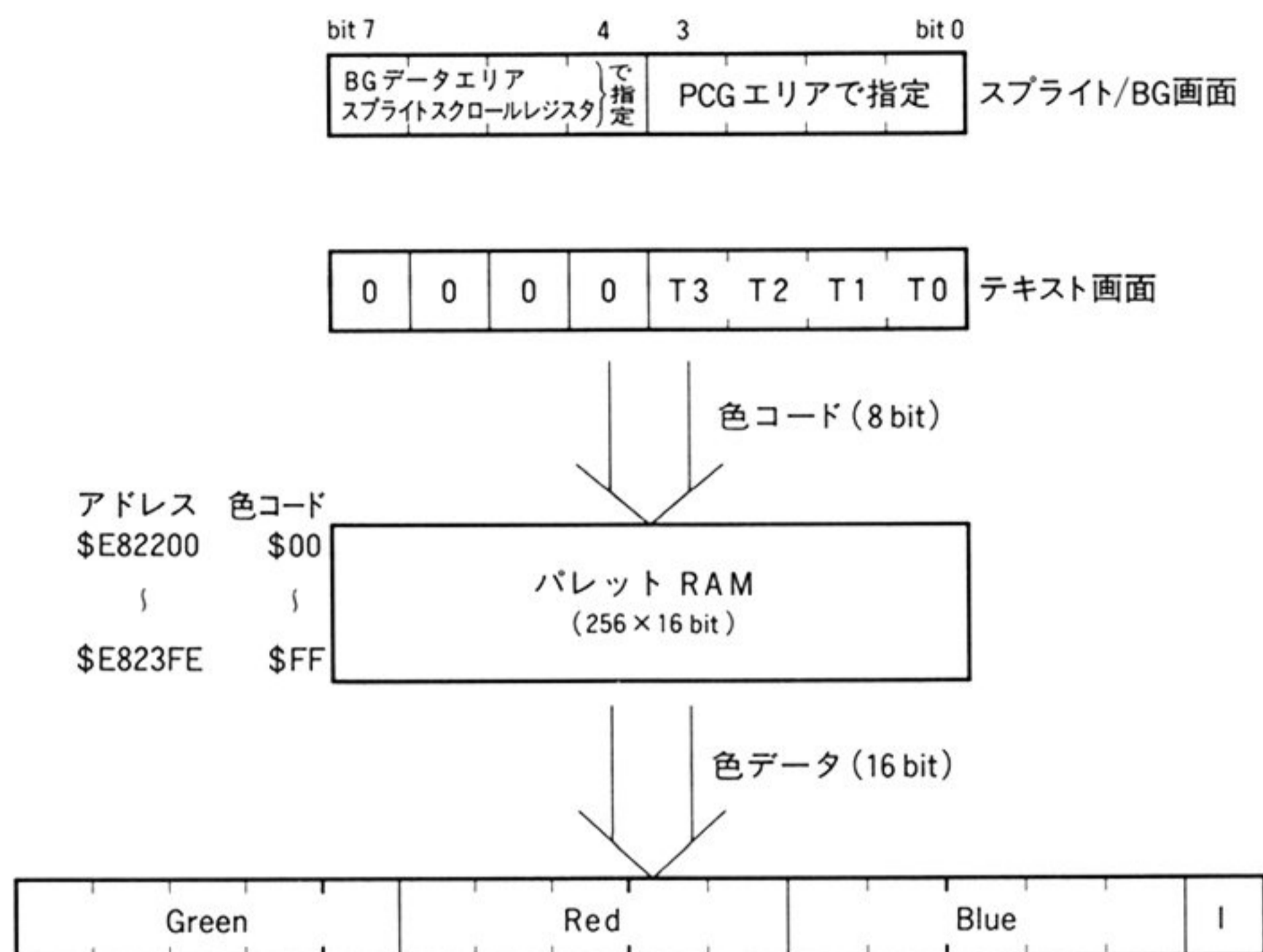
ここではまず，構造のかんたんなテキストとスプライト＋BG 画面用のパレットについて説明した後，グラフィック画面用のパレットについて説明することにします。

❸·❻1 テキストパレット

1 テキストパレット機構

テキストパレットの機構を 214 ページの図 33 に示します。16 ビット長のパレット RAM が 256 ワード分あり，テキスト画面やスプライト＋BG 画面から入力される色コードによって，この中の 1 つが選択され，そこに書き込まれている 16 ビットデータが色データとして出力されます。

スプライト＋BG 画面では，色データの下位 4 ビットは PCG エリアで，上位 4 ビットはそれぞれスプライトスクロールレジスタや BG データエリアで指定されて計 8 ビットのデータとなります。一方，テキスト画面は 4 つのプレーンがそれぞれ色コードの下位 4 ビットに対応します。上位 4 ビットはつねに 0 として扱われ，色コードの 0～15 までのパレットが使用されることになります。

●図……33　テキストとスプライト用＋BG画面パレット機構

2 テキストパレットのアドレス配置

テキストパレットRAMのアドレス配置を215ページの図34に示します。テキストパレットは$E82200〜$E823FFの512バイトに割り付けられています。各パレットは16ビット長あり，色コードが0のときには$E82200番地の16ビットデータが，1のときには$E82202番地のデータが出力されます。出力される16ビットの色データは，ビット0が輝度ビット，ビット1〜5がBlue，ビット6〜10がRed，ビット11〜15がGreenの成分になります。

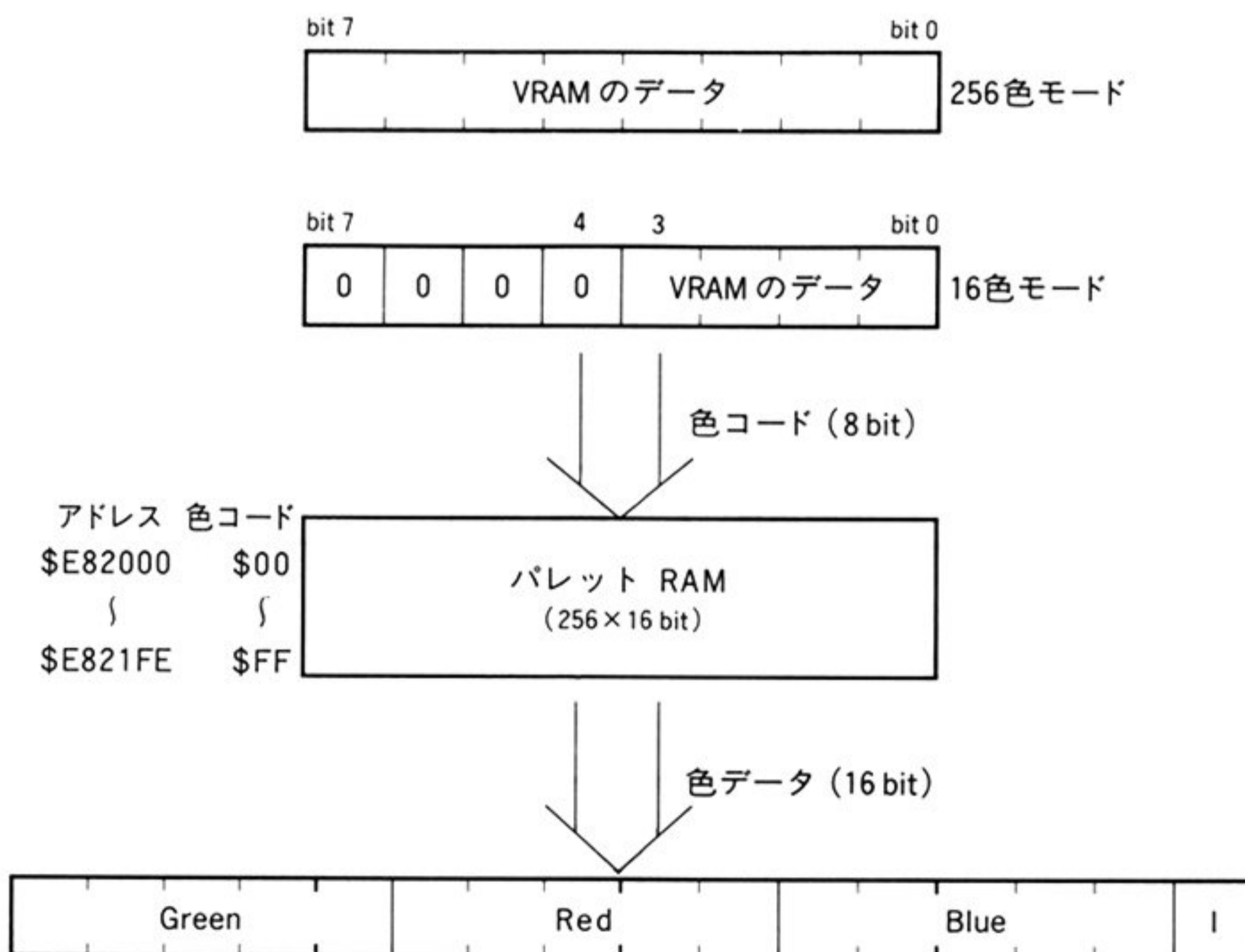
❸・❻ 2 グラフィックパレット

1 グラフィックパレット機構

グラフィックパレットは，16/256色モードのときと，65536色のときとで大きく構造が変化します。16/256色モードのときのパレットの機構を215ページの図35に，65536色モードのときの機構を図36に示します。16/256色モードのときのパレットの機構は，パレットアドレスが異なるほかはテキストパレットとほとんど同じです。グラフィック画面の場合，VRAMに

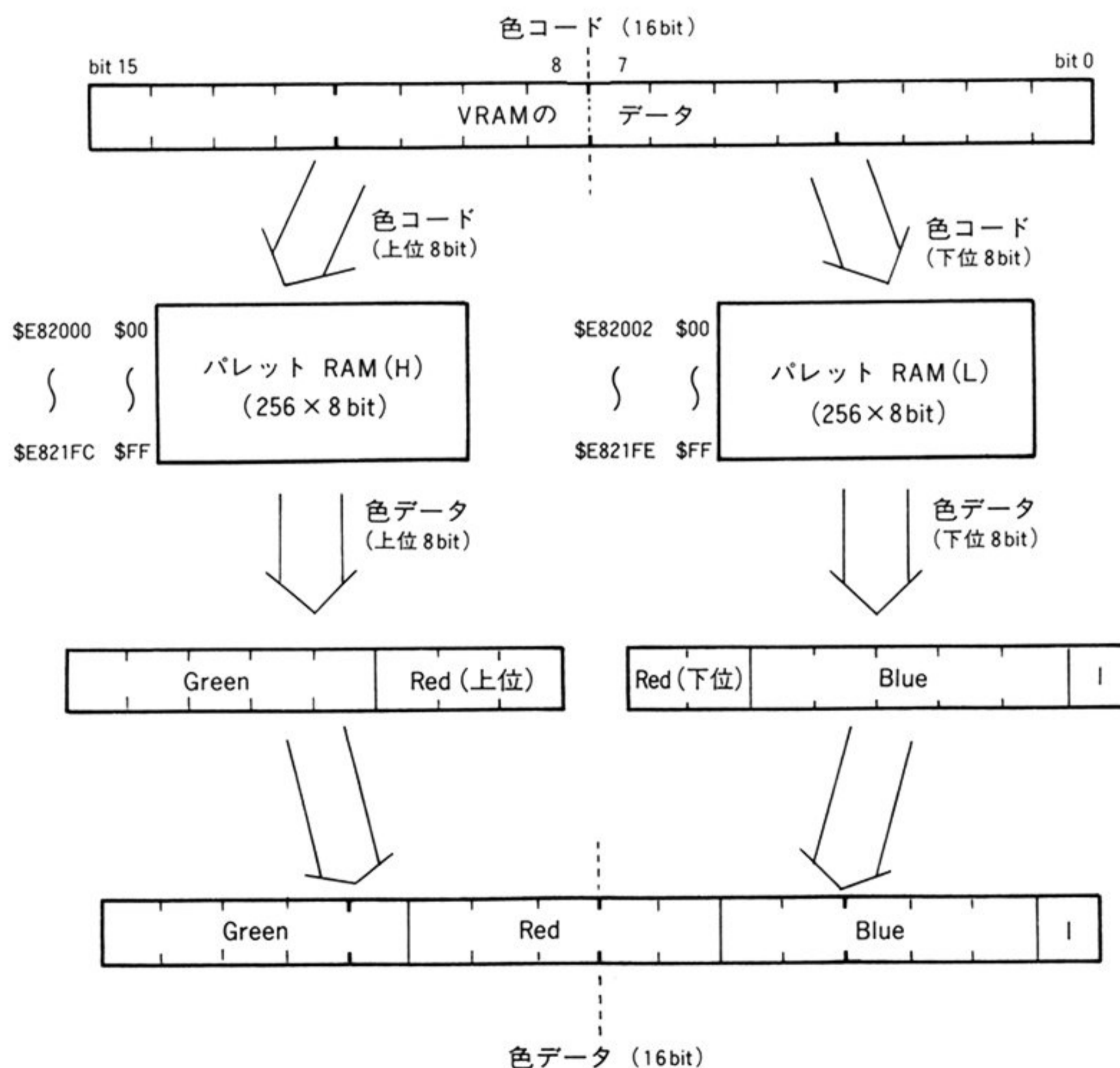
直接色コードを書き込みますが，この値がそのままパレットを選択するデータとして使用されます。

●図……34　テキスト，スプライト＋BG画面用パレット

●図……35　グラフィックパレットの機構（16/256色モード時）

65536色モード時のパレットの機構は，テキストパレットや16/256色モードのときとはずいぶん変わったものとなっています。

まず，パレットが16ビット×256個という構造であったものが，8ビット×256個×2組という構造に変化します。グラフィックVRAMから入力された16ビットの色コードは上位8ビット，下位8ビットに分割され，それぞれのコードによって2組のパレットの中から1つを選択します。そして，この2組のパレットから出力された8ビットデータが連結されて16ビットの色データとなります。

65536色モード時のパレットはこのような構造になっているため，パレットの内容を1つ書き換えると，256色分に影響してしまいます。たとえば，色コードが$0123のときの青の色が少し足りないので，該当するパレットを書き換えて青色のデータを増やすと，$0223や$0323など，色コードの下位8ビットが$23である色すべての青色が増加してしまいます。テキスト画面やグラフィックの16/256色モードでは，必ず色コードの1つ1つに色データが対応するよう

●図……36　グラフィックパレットの機構（65536色モード時）

になっているため，このようなことは起こりません。

65536色モードのときのパレットは，このように他のモードのときに比べて少々扱いにくいことや，ハード的に表示可能な色すべてを同時表示できるため，パレットを操作する意味があまり見あたらないことから，画面の初期化時に色コードと色データが等しくなるように設定されたままになっているのが普通です。

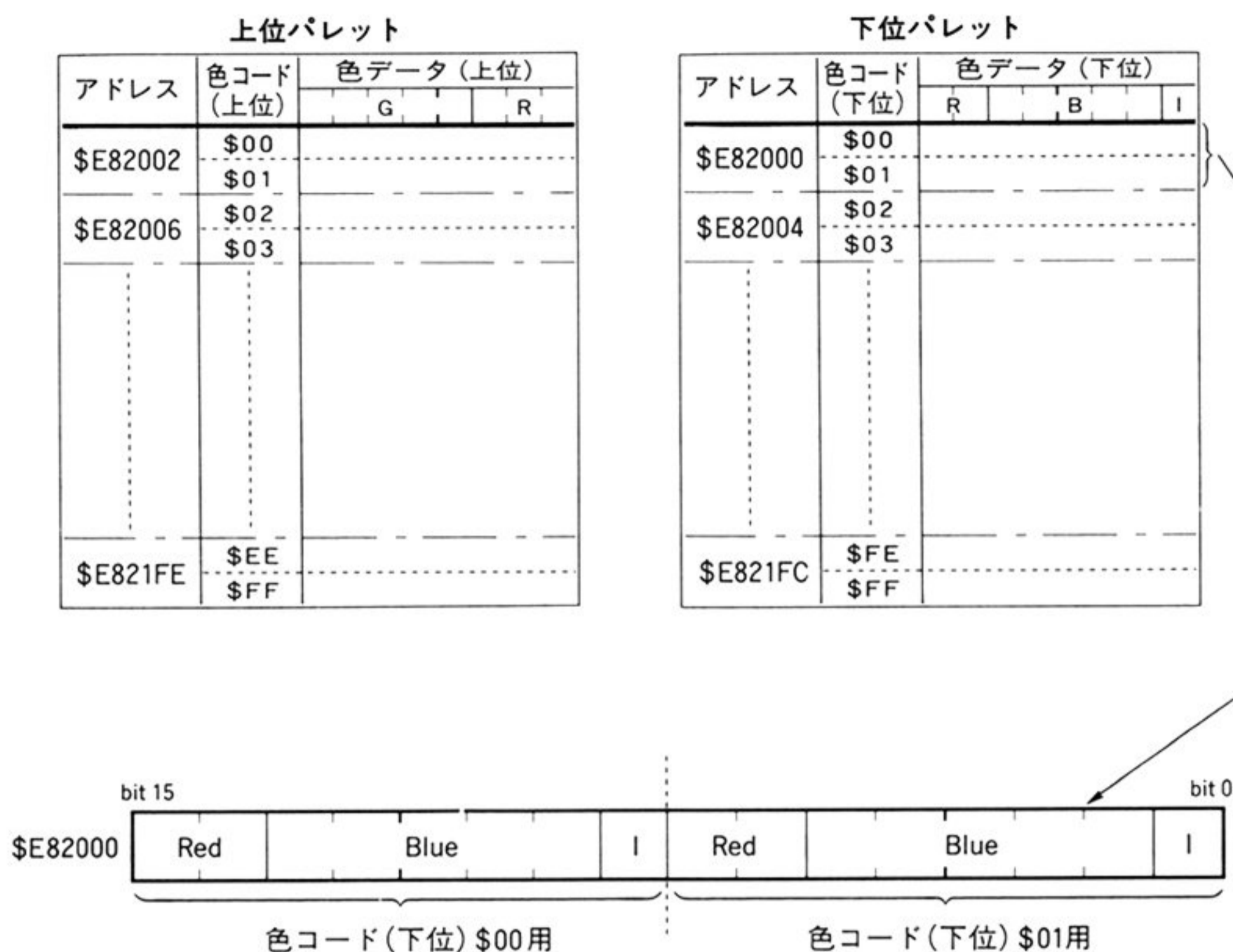
2 グラフィックパレットのアドレス配置

16/256色モードのときのパレットRAMのアドレス配置を図37に示します。パレットRAMは$E82000～$E821FFの512バイト分があり，色コードが'0'のときには$E82000番地の内容が，'1'のときには$E82002番地の内容が出力されます。出力される色データのビット配置は，テキストパレットと同様にビット0に輝度，ビット1～5にBlue，6～10にRed，11～15にGreenの成分データとして扱われます。

65536色モードのときのパレットRAMのアドレス配置を図38に示します。色コードの下位8ビットの変換用に使われるパレット（下位パレットと呼ぶことにします）は$E82000番地から4番地おきに，上位8ビットの変換に使われるパレットは$E82002番地から4番地おきに配置されています。

上位パレットから出力されるデータはGreen，およびRedの上位3ビット，下位パレットから出力されるデータはRedの下位2ビット，Blue，輝度データとなっており，連結されて得ら

●図……37　グラフィック用パレット（16/256色モード時）

●図……38　グラフィック用パレット(65536色モード時)

れる16ビットデータのビット配置は16/256色モード時のグラフィックパレットやテキストパレットと同一です。

パレットRAMの配置は，色コードが偶数のときのデータと奇数のときのデータをまとめて1ワード (16ビット) としてアクセスできるようになっています。たとえば，$E82000番地の16ビットデータの上位8ビットには色コードの下位8ビットが$00のときの色データ (正確にはデータの下位8ビット)が，下位8ビットには色コードの下位8ビットが$01のときの色データが設定されます。

4 CGROM(キャラクタジェネレータROM)

CGROMは，英数字や漢字の文字パタン (以下，フォントと呼びます) が書き込まれている

ROMのことです。X 68000は，テキスト画面もビットマップ方式を採用しており，どのような形の文字でも表示できることから，CGROMにもさまざまな大きさの文字パタンが用意されています。あらかじめ用意されている文字パタンの一覧を図39に示します。

CGROM内には英数字(半角，1/4角)フォントとして8×8ドット，8×16ドット，12×12ドット，12×24ドットの4種類，漢字(全角)フォントとして16×16ドット，24×24ドットの2種類の計6種類のフォントがあります。Human 68 KのIOCSコールなどでサポートされているのは8×16ドットの半角文字と16×16ドットの全角文字だけですが，SX-WINDOW上では他のフォントの表示も行えるようになっています。図中，文字レターフェースとなっているのは，実際の文字の大きさです。英数字をフォントの大きさいっぱいに配置すると，密着して配置したときにたいへん見にくくなります。このため，実際の文字パターンは，文字フォントサイズとして定義されている領域よりも小さくして余った端のドットを空白にすることで，密着して配置されても読みやすくなるようにしているわけです。

CGROMのアドレス配置は図40のようになっています。$F00000～$F388BFに16×16ドットの全角フォントが，$F3A000～$F3A7FFに8×8ドット，$F3A800～$F3B7FFに8×16ドット，$F3B800～$F3CFFFに12×12ドット，$F3D000～$F3FFFFに12×24ドットの半角フォントが配置され，さらに$F40000～$FBF3AFに24×24ドットの全角フォントが格納されています。16×16ドットと8×8ドットのフォントデータ領域の間や，24×24ドットフォント領域の終わりとCGROM領域の最終アドレスである$FBFFFFまでのすき間はたんなる空き領域です。

次に，それぞれのフォントがCGROM内にどのように格納されているのかを説明することにしましょう。

●図……39　ROMで持っている文字フォント

文字種		フォントサイズ	文字 レターフェース	文字コード
英数字	1/4角文字	8×8 12×12	6×7 10×10	$00～$FF ⋮
	半角文字	8×16 12×24	7×13 10×18	
漢　字 非漢字	全角文字	16×16 24×24	15×16 24×24	非 漢 字：JISコード上位$21～$28 第一水準：　〃　$30～$4F 第二水準：　〃　$50～$74 下位 $21～$7E
		(ヨコ×タテ)	(ヨコ×タテ)	

文字種数
- 英 数 字：256
- 第一水準：3008
- 第二水準：3478

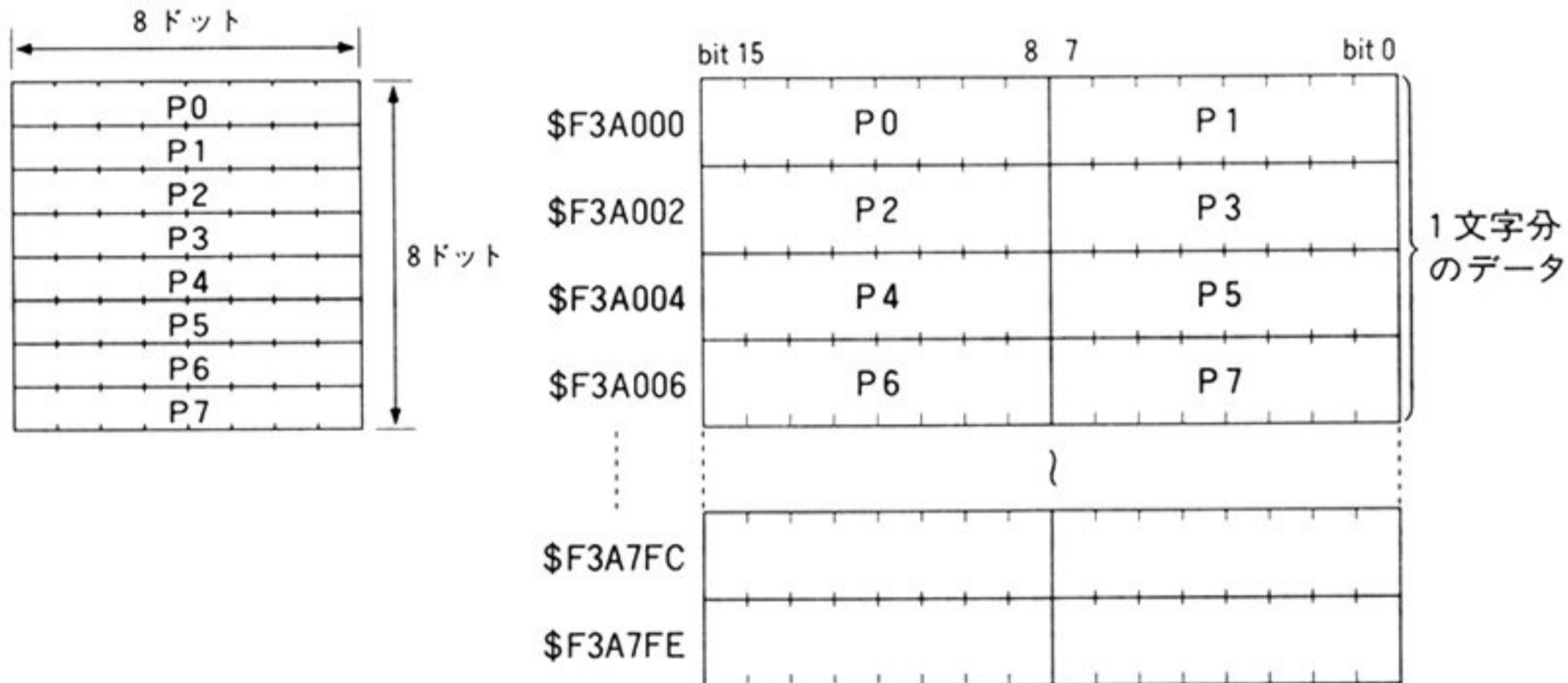
●図……40　CGROMアドレス配置

❹·1 8×8ドットフォント

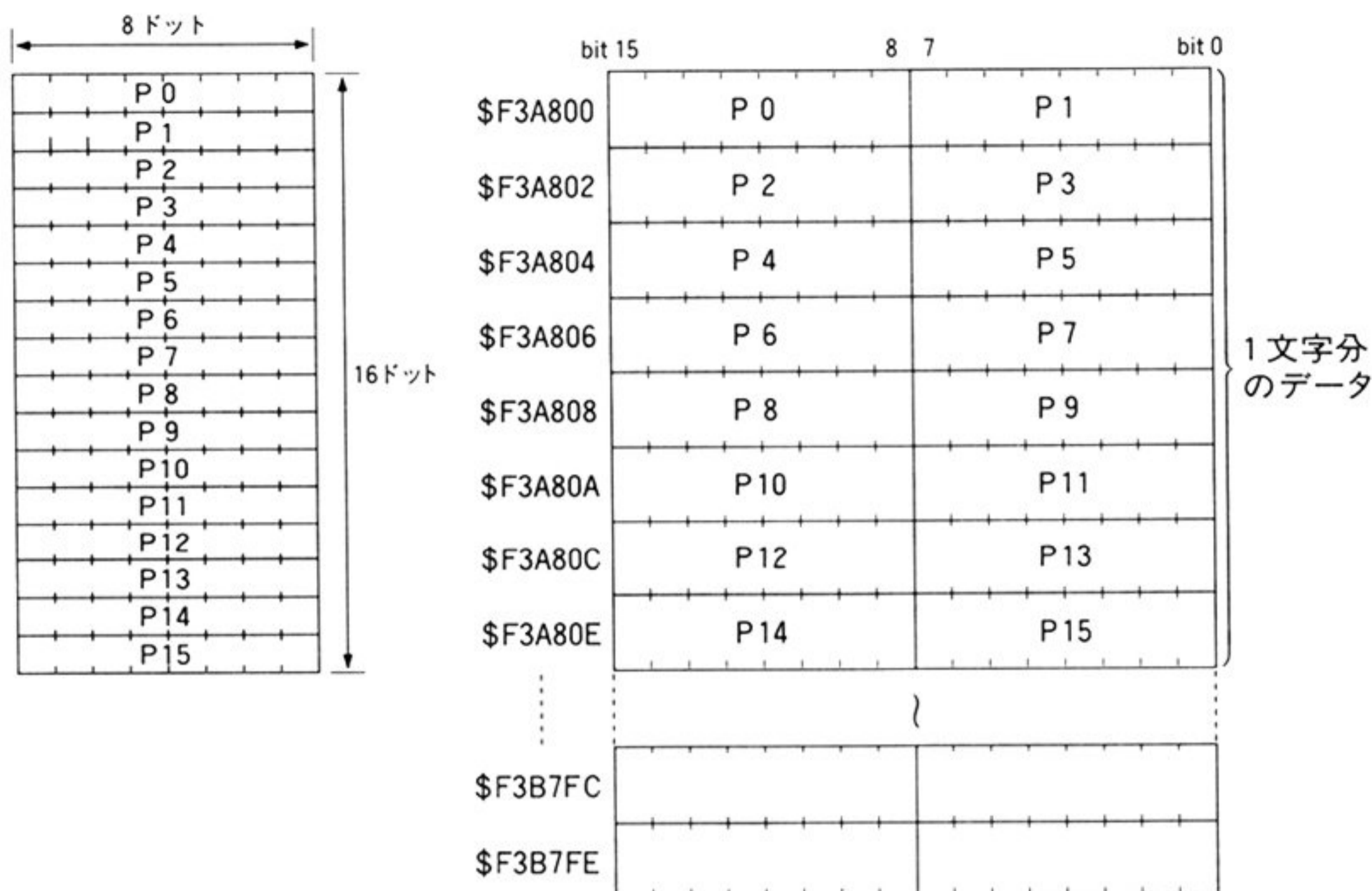
8×8ドットフォントデータの格納のされ方を図41に示します。フォントデータは$F3A000から始まり，1文字あたり8バイト分のメモリ領域を使用しています。

文字フォントは横8ドットが1バイトで表現されており，8バイトで8×8ドット分のパタンになります。水平8ドットは右端がビット0，左端がビット7に対応します。

CPUからCGROMへのアクセスは，バイト単位でもワード単位でも可能です。ワード(16ビット単位)アクセスでCGROMを読み出したときには，いちばん上のラインのパタンデータが上位8ビット，次のラインのデータが下位8ビットになります。これは通常のメモリアクセスのときと同じことなので，とくに気にすることはないでしょう。

●図……41 8×8ドットフォント

❹·2 8×16ドットフォント

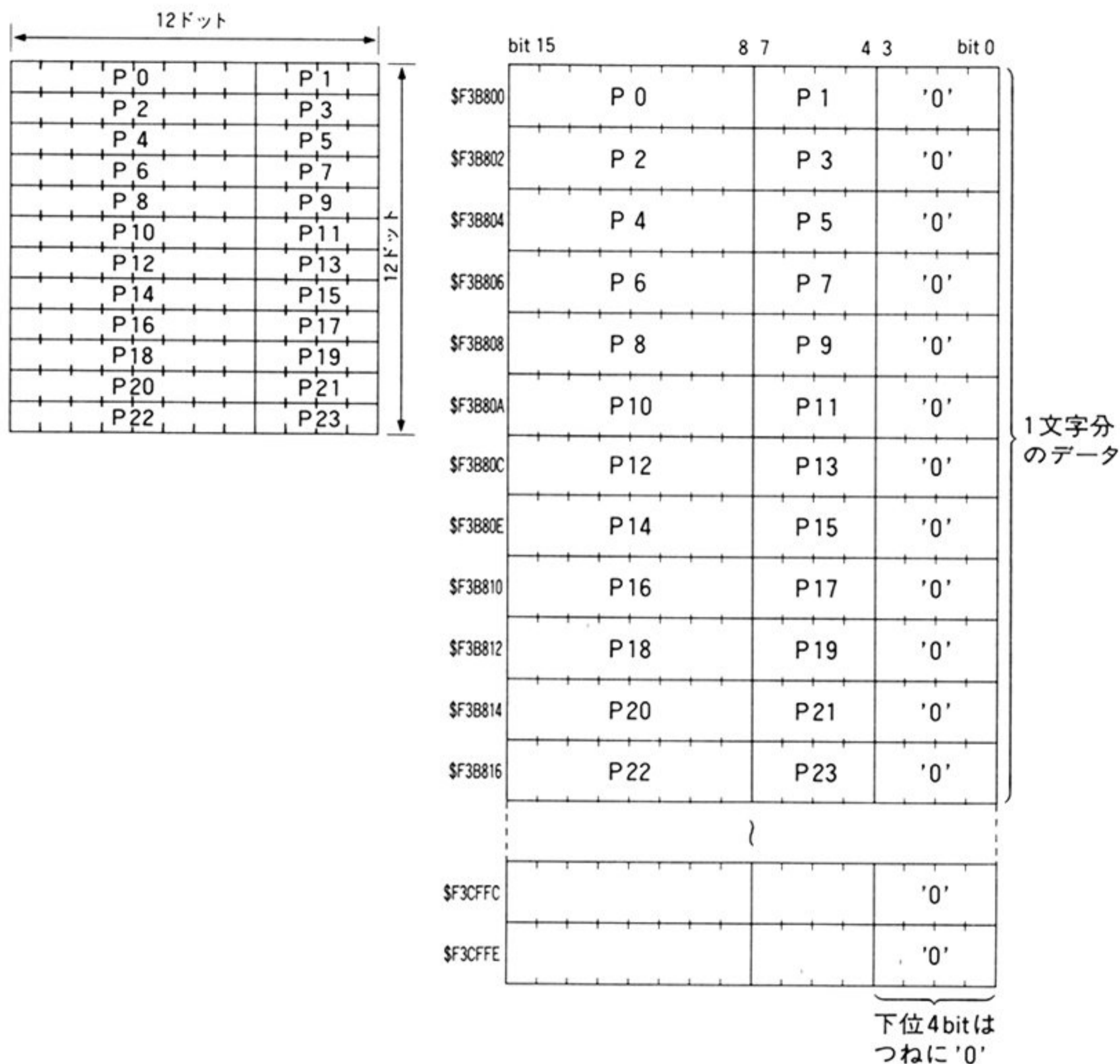
8×16ドットフォントデータの格納のされ方を図42に示します。開始アドレスが$F3A800から始まり，1文字あたり16バイトを使用していること以外は8×8ドットフォントと同じです。

●図……42　8×16ドットフォント

❹·3 12×12ドットフォント

12×12ドットフォントデータは223ページの図43のように格納されています。開始アドレスは$F3B800で，水平1ラインに2バイト，1文字あたり24バイトを使用しています。

8ビット単位でメモリにアクセスするCPUにとって，12という数値は中途半端です。CGROMでは，1ラインに2バイト（16ビット）分の領域を使い，このうち上位12ビットにパタンを登録しています。残った下位4ビットはすべて'0'が読み出されます。

1ワード（16ビット）単位で読み出したときには，パタンの右端はビット4，左端はビット15となります。

●図……43　12×12 ドットフォント

❹·4　12×24ドットフォント

12×24 ドットフォントデータは図 44 のように格納されています。開始アドレスは$F3D 000，水平 1 ラインは 2 バイト，1 文字あたり 48 バイトを使用しています。

水平データの構造は，12×12 ドットフォントのときと同じように，1 ラインに 2 バイト(16 ビット)を使い，このうち上位 12 ビットにパタンが登録されています。下位 4 ビットは，12×12 ドットと同様，つねに'0'となっています。

●図……44　12×24 ドットフォント

12ドット

P 0	P 1
P 2	P 3
P 4	P 5
P 6	P 7
P 8	P 9
P10	P11
P12	P13
P14	P15
P16	P17
P18	P19
P20	P21
P22	P23
P24	P25
P26	P27
P28	P29
P30	P31
P32	P33
P34	P35
P36	P37
P38	P39
P40	P41
P42	P43
P44	P45
P46	P47

24ドット

	bit 15 … 8	7 … 4	3 … bit 0
$F3D000	P 0	P 1	'0'
$F3D002	P 2	P 3	'0'
$F3D004	P 4	P 5	'0'
$F3D006	P 6	P 7	'0'
$F3D008	P 8	P 9	'0'
$F3D00A	P10	P11	'0'
$F3D00C	P12	P13	'0'
$F3D00E	P14	P15	'0'
$F3D010	P16	P17	'0'
$F3D012	P18	P19	'0'
$F3D014	P20	P21	'0'
$F3D016	P22	P23	'0'
$F3D018	P24	P25	'0'
$F3D01A	P26	P27	'0'
$F3D01C	P28	P29	'0'
$F3D01E	P30	P31	'0'
$F3D020	P32	P33	'0'
$F3D022	P34	P35	'0'
$F3D024	P36	P37	'0'
$F3D026	P38	P39	'0'
$F3D028	P40	P41	'0'
$F3D02A	P42	P43	'0'
$F3D02C	P44	P45	'0'
$F3D02E	P46	P47	'0'
～			
$F3FFFC			'0'
$F3FFFE			'0'

1文字分のデータ

下位4 bitは
つねに'0'

❹·5　16×16ドットフォント

16×16 ドットフォントのデータは 225 ページの図 45 のように格納されています。開始アドレスは$F00000，水平１ラインは２バイト，１文字あたり 32 バイトを使用しています。

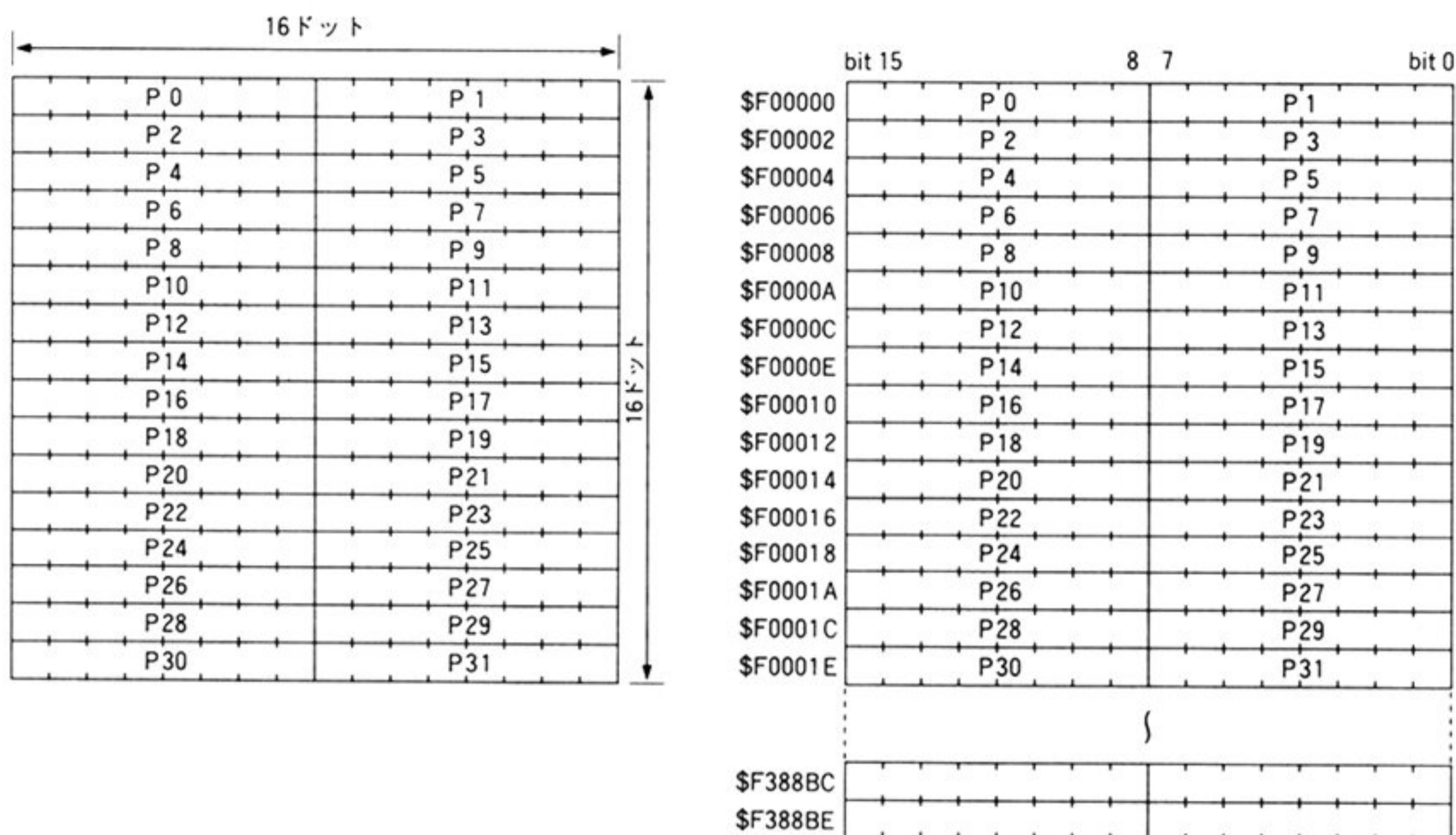
文字フォントの水平 16 ドットは，そのまま１ワードのデータとして格納されています。ビット配置は右端がビット０，左端がビット 15 になっています。

●図……45　16×16 ドットフォント

16ドット

P 0	P 1
P 2	P 3
P 4	P 5
P 6	P 7
P 8	P 9
P10	P11
P12	P13
P14	P15
P16	P17
P18	P19
P20	P21
P22	P23
P24	P25
P26	P27
P28	P29
P30	P31

16ドット

	bit 15 … 8	7 … bit 0
$F00000	P 0	P 1
$F00002	P 2	P 3
$F00004	P 4	P 5
$F00006	P 6	P 7
$F00008	P 8	P 9
$F0000A	P10	P11
$F0000C	P12	P13
$F0000E	P14	P15
$F00010	P16	P17
$F00012	P18	P19
$F00014	P20	P21
$F00016	P22	P23
$F00018	P24	P25
$F0001A	P26	P27
$F0001C	P28	P29
$F0001E	P30	P31
∫		
$F388BC		
$F388BE		

❹·6　24×24ドットフォント

24×24 ドットフォントデータは図 46 のように格納されています。開始アドレスは$F40000 で，水平 1 ライン 24 ドット分が 3 バイト，1 文字あたり 72 バイトを使用します。

1 ライン分のデータは 24 ビットとなっていますが，CPU のメモリアクセスはバイト（8 ビット），ワード（16 ビット），ロングワード（32 ビット）が基本であるため，24 ビットのアクセスはバイト単位でのアクセスを 3 回繰り返す，16 ビット＋8 ビットと 2 回に分ける，32 ビット分を読み出して 8 ビット分を切り捨てるなどの工夫が必要です（奇数番地からワード単位やロングワード単位のアクセスを行おうとするとアドレスエラーが発生するので，プログラムを組むときには注意してください）。

●図……46　24×24 ドットフォント

24ドット（横）× 24ドット（縦）

P 0	P 1	P 2
P 3	P 4	P 5
P 6	P 7	P 8
P 9	P10	P11
P12	P13	P14
P15	P16	P17
P18	P19	P20
P21	P22	P23
P24	P25	P26
P27	P28	P29
P30	P31	P32
P33	P34	P35
P36	P37	P38
P39	P40	P41
P42	P43	P44
P45	P46	P47
P48	P49	P50
P51	P52	P53
P54	P55	P56
P57	P58	P59
P60	P61	P62
P63	P64	P65
P66	P67	P68
P69	P70	P71

	bit 15 … 8	7 … bit 0
$F40000	P 0	P 1
$F40002	P 2	P 3
$F40004	P 4	P 5
$F40006	P 6	P 7
$F40008	P 8	P 9
$F4000A	P10	P11
$F4000C	P12	P13
$F4000E	P14	P15
$F40010	P16	P17
$F40012	P18	P19
$F40014	P20	P21
$F40016	P22	P23
$F40018	P24	P25
$F4001A	P26	P27
$F4001C	P28	P29
$F4001E	P30	P31
$F40020	P32	P33
$F40022	P34	P35
$F40024	P36	P37
$F40026	P38	P39
$F40028	P40	P41
$F4002A	P42	P43
$F4002C	P44	P45
$F4002E	P46	P47
$F40030	P48	P49
$F40032	P50	P51
$F40034	P52	P53
$F40036	P54	P55
$F40038	P56	P57
$F4003A	P58	P59
$F4003C	P60	P61
$F4003E	P62	P63
$F40040	P64	P65
$F40042	P66	P67
$F40044	P68	P69
$F40046	P70	P71
⋮		
$FBF3AC		
$FBF3AE		

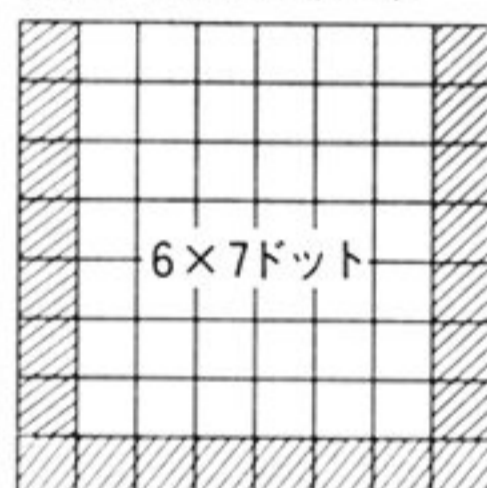
C O L U M N

CGROMのパタン配置の実際

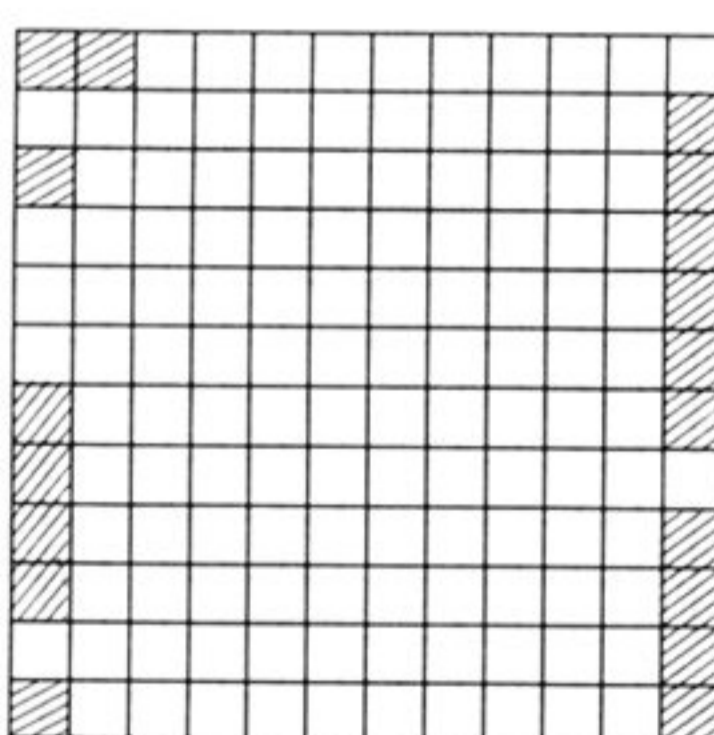
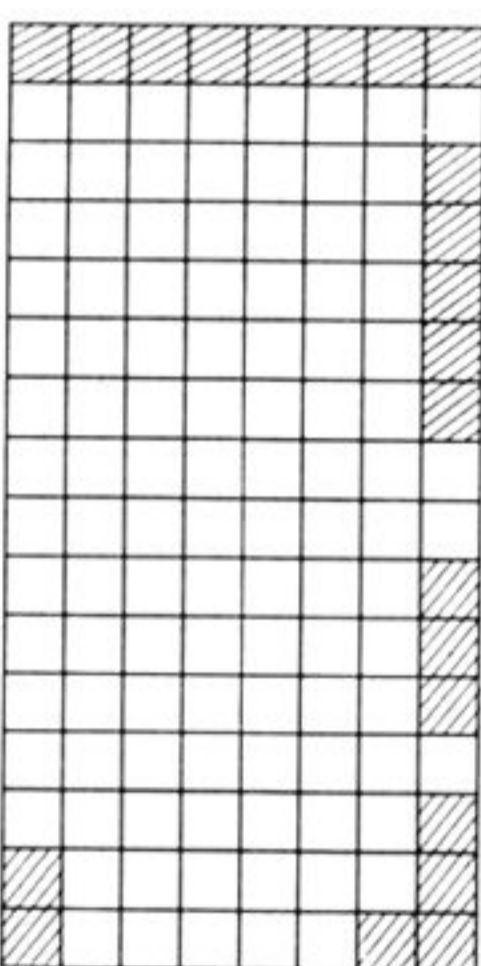
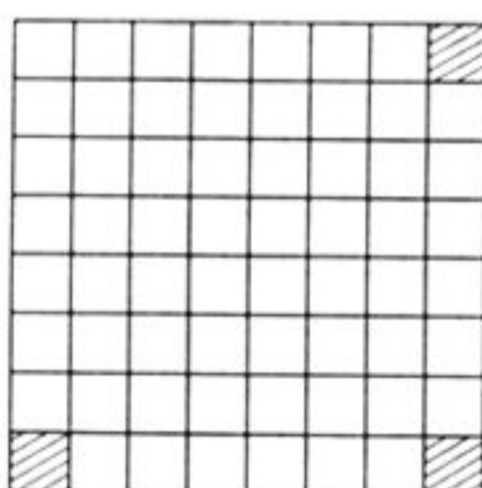
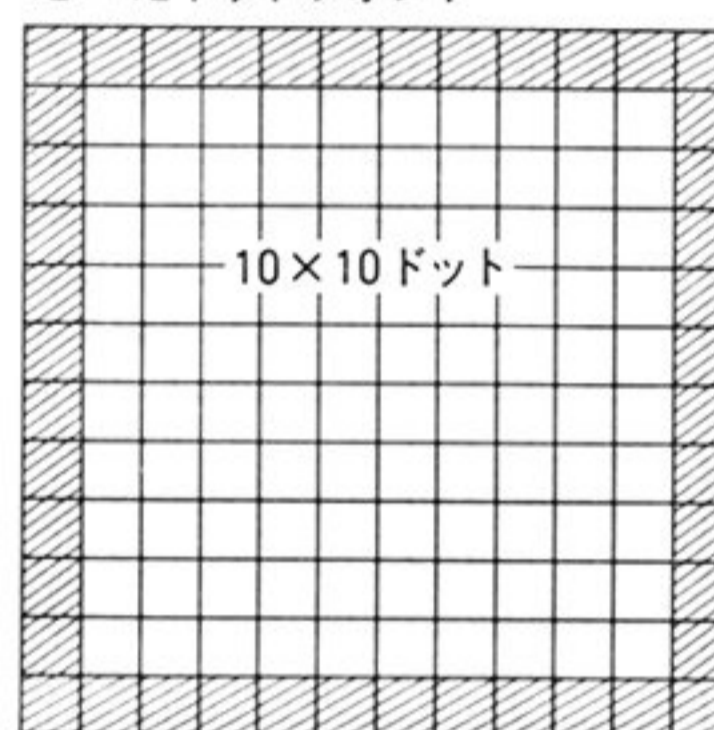
本文中では，パタンの実際の配置がどうなっているかについては触れなかったので，ここで補足しておきます。なお，このフォントの配置は筆者が個人的に調べただけですので，将来にわたって変更されない保証はありません。

公開されているレターフェースの値と実際の CGROM の内容から，各フォントが極力めざしていると思われるパタン配置領域と，実際にパタンが配置されている領域を調べてみたのが図 47〜図 49 です。いずれも上段がレターフェースの領域，下段が実際にパタンが配置されている領域です。ずいぶんレターフェース領域をはみ出していることがわかると思います。よく調べてみると，アルファベットの大文字や数字などはほとんどがレターフェース領域の中に配置されているのですが，かな文字や記号の一部などがレターフェース領域をはみ出しているようです。

また，16×16 ドットフォント，24×24 ドットフォントのレターフェースはそれぞれ 15×16，24×24 となっていますが，アルファベットの大文字や数字などは図に点線で書いた 15×13 ド

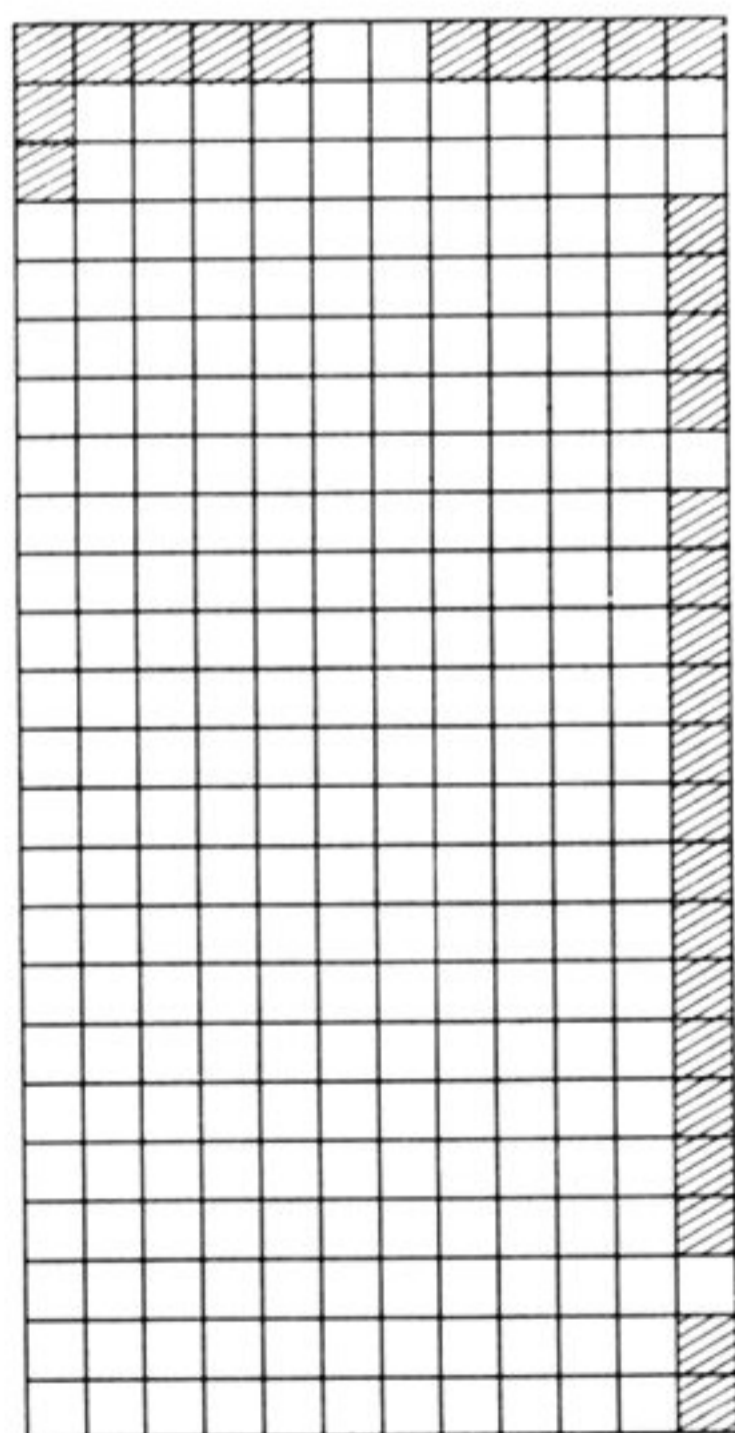
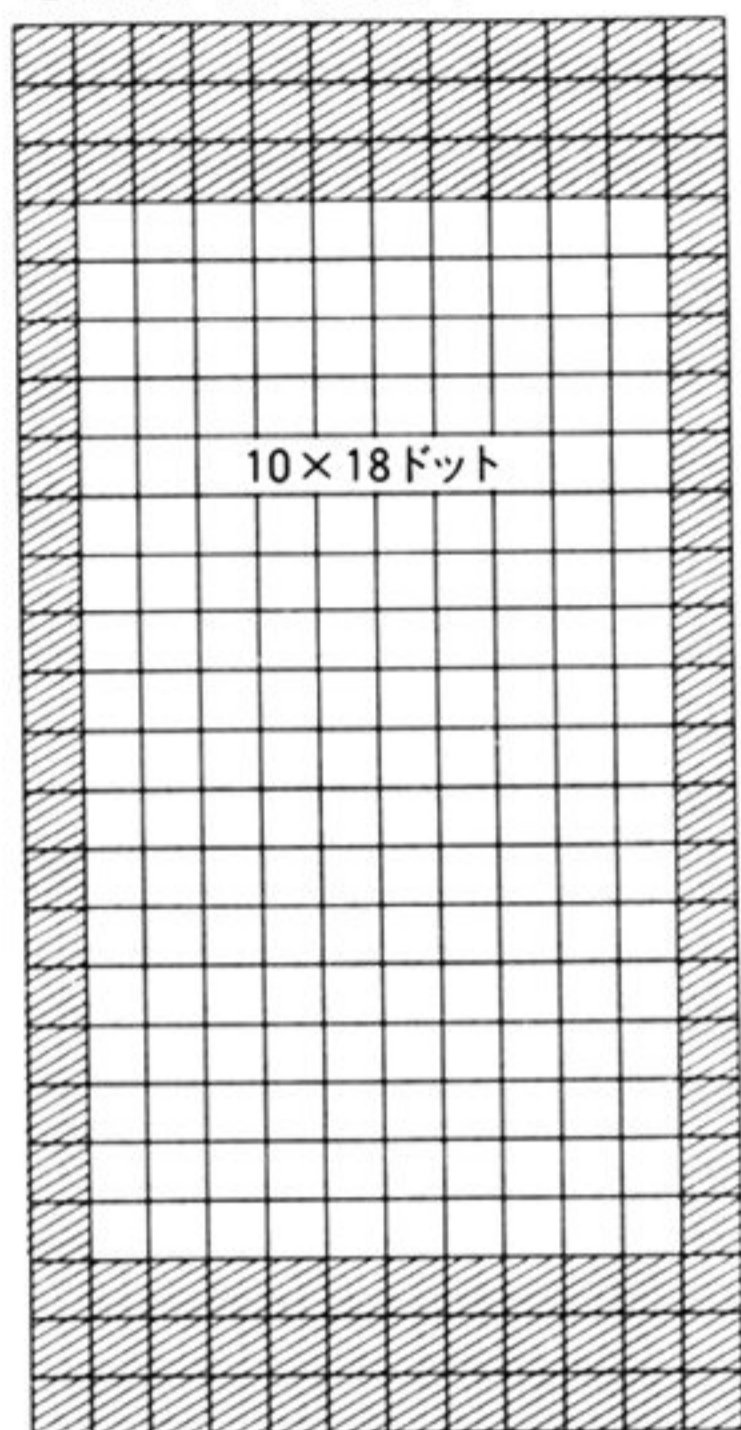
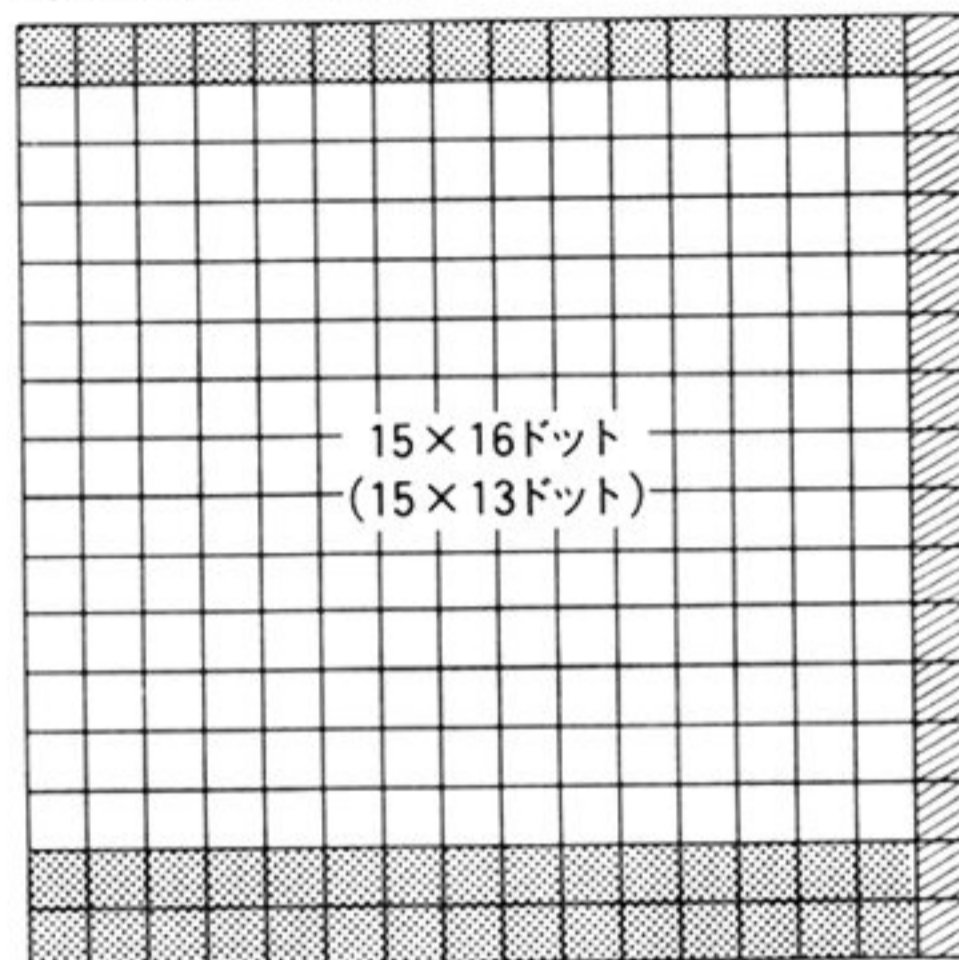
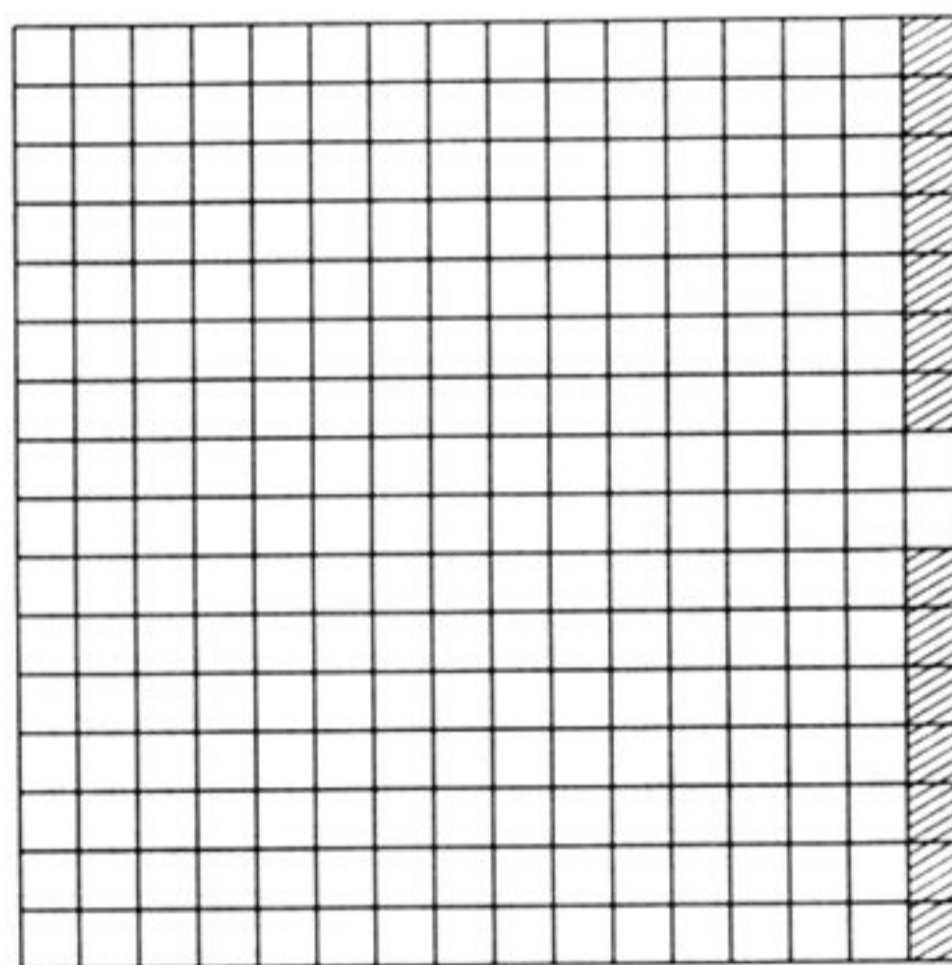
●図……47　パタン使用領域(1)

8×8 ドットフォント

8×16ドットフォント

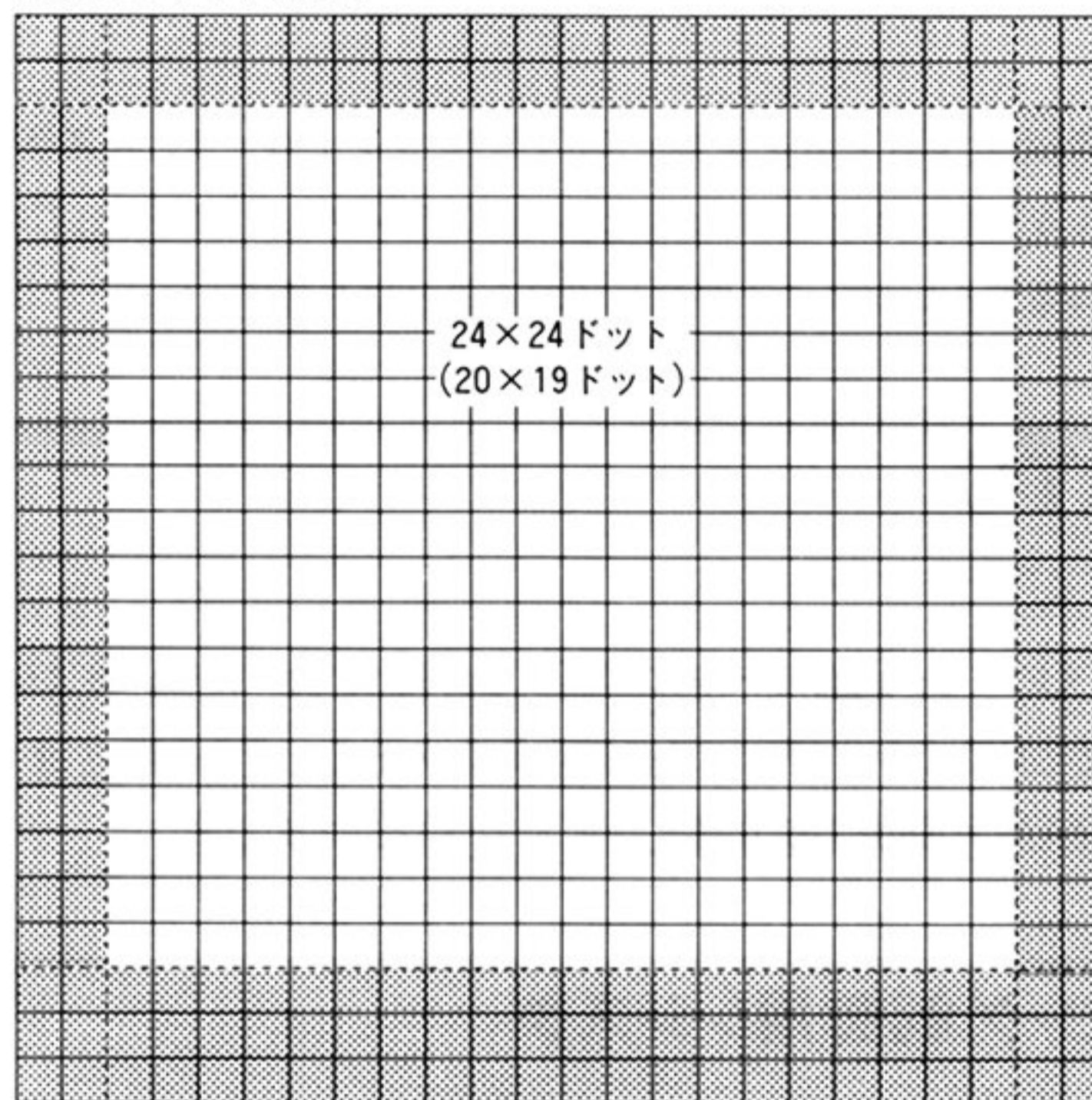
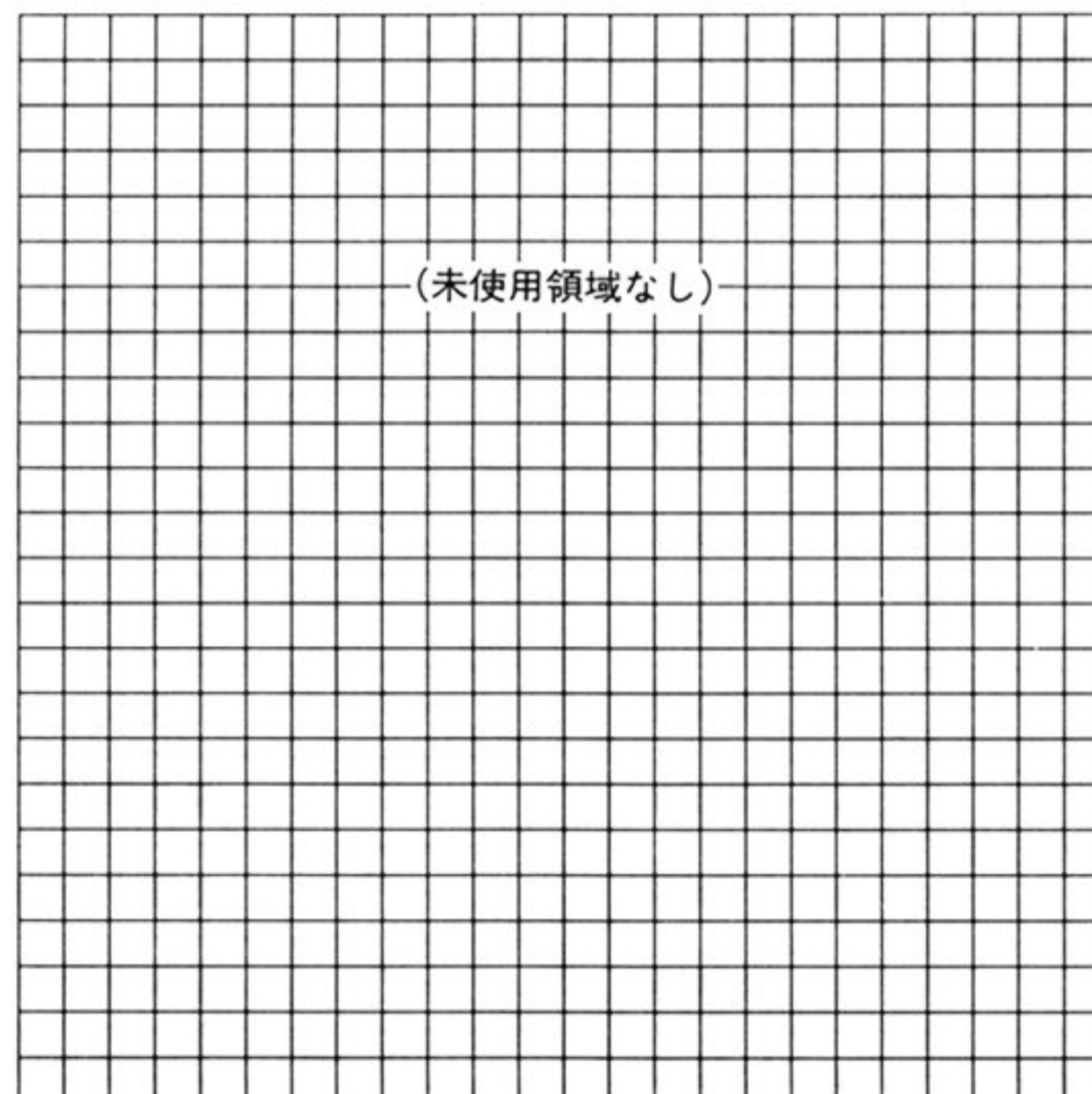
12×12 ドットフォント

上段：レターフェース領域
下段：実際のパタン配置領域

ット，20×19 ドットの領域に配置されているようです。密着配置された場合を考慮したか，漢

●図……48　パタン使用領域(2)

12×24ドットフォント

16×16ドットフォント

( )内はアルファベット大文字数字のレターフェース

字とのバランスをとるためではないかと思われます。

●図……49　パタン使用領域(3)

24×24ドットフォント

(　)内はアルファベット大文字，英数字のレターフェース

●5 画面モード制御

X 68000の画面表示は，CRT コントローラ，ビデオコントローラ，スプライトコントローラの協調作業で行われるため，画面モードの設定に多くのレジスタが関係しています。ここでは，画面モード設定に関係するレジスタを整理しておくことにします。

❺·1 CRTC

CRTCのレジスタのうち，画面モードに関係するのはR 00～R 08，およびR 20です。このうちR 00～R 07はCRT インタフェースの基本的なタイミングの調整を行うレジスタ，R 20は色モードなど VRAM の構成の切り替えなどを行うレジスタです。

❺·❶1 タイミング制御用レジスタ

R 00～R 08の配置は図 50のようになっています。

●図……50 CRTC(R00～R08)

			bit 15 … 8 7 … bit 0
水平タイミング制御	R00	$E80000	水平トータル*'1'
	R01	$E80002	水平同期終了位置
	R02	$E80004	水平表示開始位置
	R03	$E80006	水平表示終了位置
垂直タイミング制御	R04	$E80008	垂直トータル
	R05	$E8000A	垂直同期終了位置
	R06	$E8000C	垂直表示開始位置
	R07	$E8000E	垂直表示終了位置
水平位置微調整	R08	$E80010	外部同期水平アジャスト

＊水平トータル値はつねに奇数(最下位ビット＝'1')にすること

これらのレジスタでは，CRTCに与える同期信号や，画面の表示期間などのタイミングの調整を行います。X 68000のCRTインタフェースの標準的なタイミングと，各画面モードでのR 00〜R 08の設定値を図51に，各タイミング名と信号波形との関係を232ページの図52に示します。このレジスタの設定値の計算方法は233ページの図53のようになっています。ただし，R 00には必ず奇数（最下位ビットを'1'にする）の値を設定するようにしてください。

❺·❶2 CRTC R20

CRTCのR 20のビット配置を233ページの図54に示します。このレジスタの上位バイトで実画面サイズと色モードを，下位バイトで水平偏向周波数（高解像度モードか，標準解像度モードか）と表示画面の垂直，水平ドット数の設定を行います。このうち，上位バイトのビット8，9，10はビデオコントローラのR 0の下位3ビットと同じ値を設定するようにしてください。

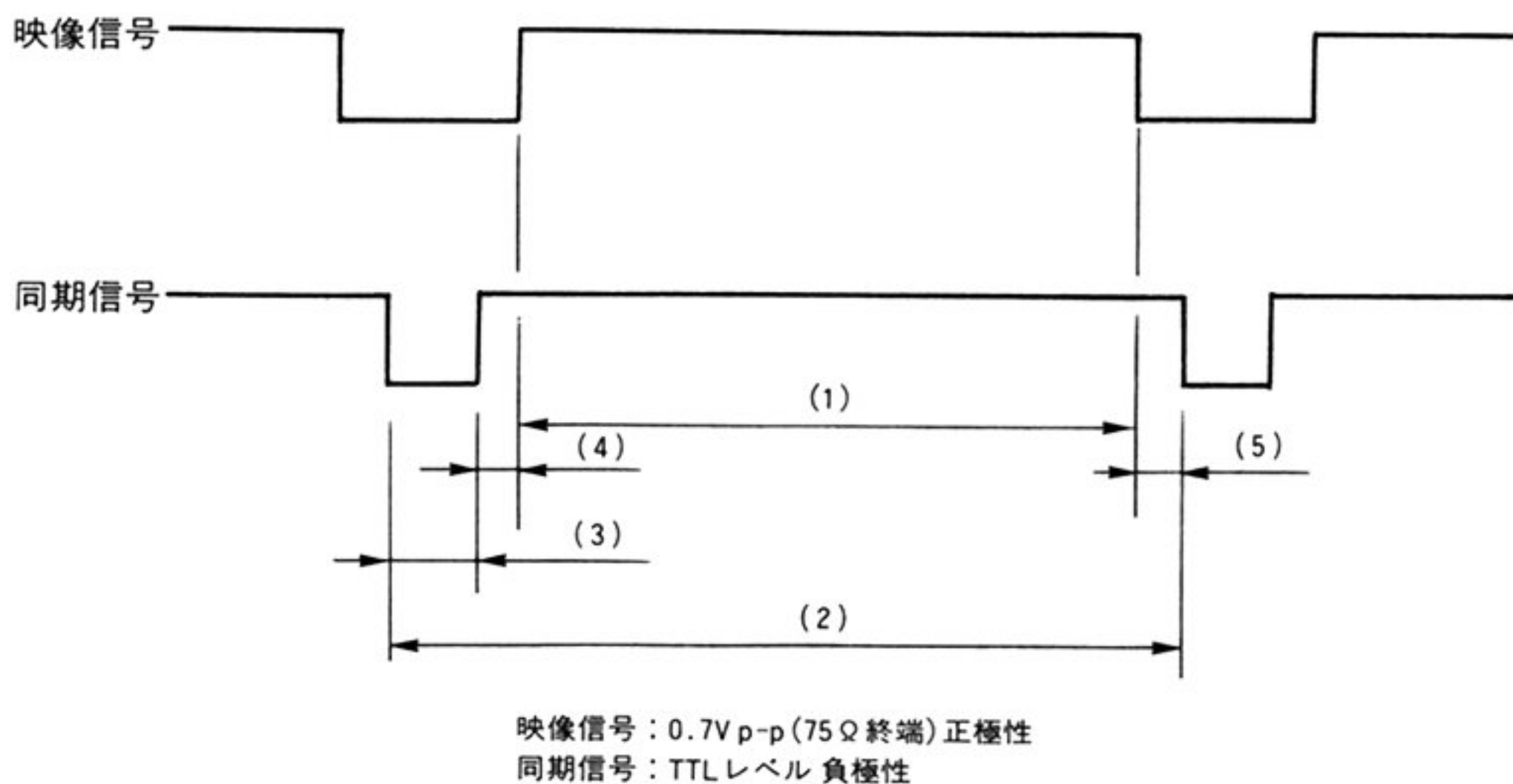
●図……51　CRTインタフェースの基本タイミングとCRTCの標準設定値

タイミング		高解像度	標準解像度
同期周波数	水平	31.5kHz	15.98kHz
	垂直	55.46Hz	61.46Hz
データ表示期間（1）	水平	22.09μs	52.69μs
	垂直	16.25ms	15.019ms
同期期間（2）	水平	31.75μs	62.58μs
	垂直	18.03ms	16.270ms
同期パルス幅（3）	水平	3.45μs	3.30μs
	垂直	0.191ms	0.187ms
バックポーチ（4）	水平	4.14μs	4.94μs
	垂直	1.111ms	0.876ms
フロントポーチ（5）	水平	2.07μs	1.65μs
	垂直	0.476ms	0.187ms

レジスタ		画面モード(高解像度)				画面モード(標準解像度)		
番号	アドレス	768×512	512×512	512×256	256×256	512×512	512×256	256×256
R 00	$E80000	$89 (137)	$5B (91)	$5B (91)	$2D (45)	$4B (75)	$4B (75)	$25 (37)
R 01	$E80002	$0E (14)	$09 (9)	$09 (9)	$04 (4)	$03 (3)	$03 (3)	$01 (1)
R 02	$E80004	$1C (28)	$11 (17)	$11 (17)	$06 (6)	$05 (5)	$05 (5)	$00 (0)
R 03	$E80006	$7C (124)	$51 (81)	$51 (81)	$26 (38)	$45 (69)	$45 (69)	$20 (32)
R 04	$E80008	$237 (567)	$237 (567)	$237 (567)	$237 (567)	$103 (259)	$103 (259)	$103 (259)
R 05	$E8000A	$05 (5)	$05 (5)	$05 (5)	$05 (5)	$02 (2)	$02 (2)	$02 (2)
R 06	$E8000C	$28 (40)	$28 (40)	$28 (40)	$28 (40)	$10 (16)	$10 (16)	$10 (16)
R 07	$E8000E	$228 (552)	$228 (552)	$228 (552)	$228 (552)	$100 (256)	$100 (256)	$100 (256)
R 08	$E80010	$1B (27)	$1B (27)	$1B (27)	$1B (27)	$2C (44)	$2C (44)	$24 (36)

(　)内は10進数

●図……52　CRT への信号

映像信号：0.7V p-p(75Ω終端)正極性
同期信号：TTLレベル 負極性

●図……53　CRTC R00～R07の設定値の算出法

$$〔R00〕=\frac{(\text{水平同期期間})\times(\text{水平表示ドット数})}{(\text{データ表示期間})\times 8}-1$$

$$〔R01〕=\frac{(\text{水平同期パルス幅})\times(\text{水平表示ドット数})}{(\text{データ表示期間})\times 8}-1$$

$$〔R02〕=\frac{\langle\!\langle(\text{水平同期パルス幅})+(\text{水平バックポーチ})\rangle\!\rangle\times(\text{水平表示ドット数})}{(\text{データ表示期間})\times 8}-5$$

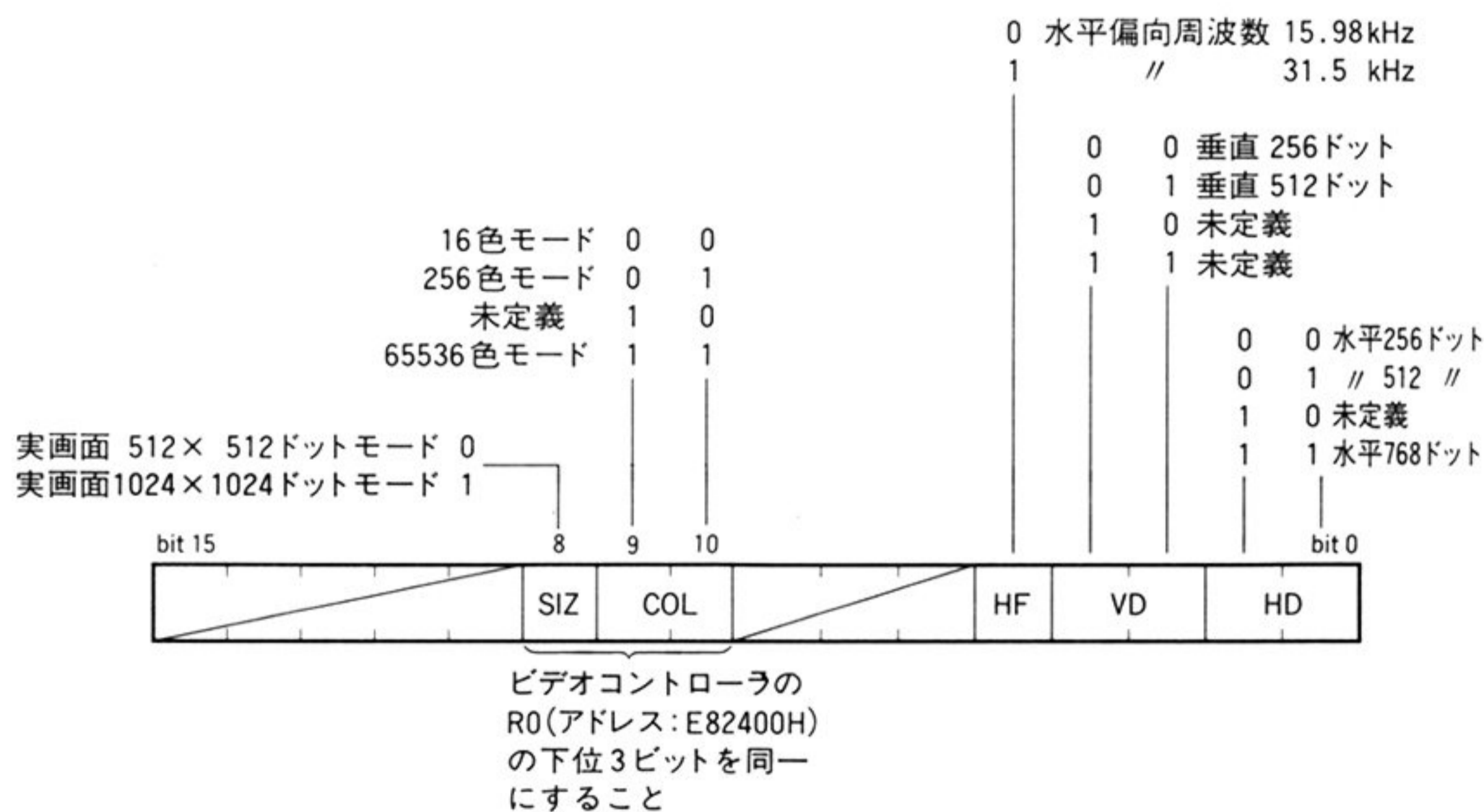
$$〔R03〕=\frac{\langle\!\langle(\text{水平同期期間})+(\text{水平フロントポーチ})\rangle\!\rangle\times(\text{水平表示ドット数})}{(\text{データ表示期間})\times 8}-5$$

$$〔R04〕=\frac{(\text{垂直同期期間})}{(\text{水平同期期間})}-1$$

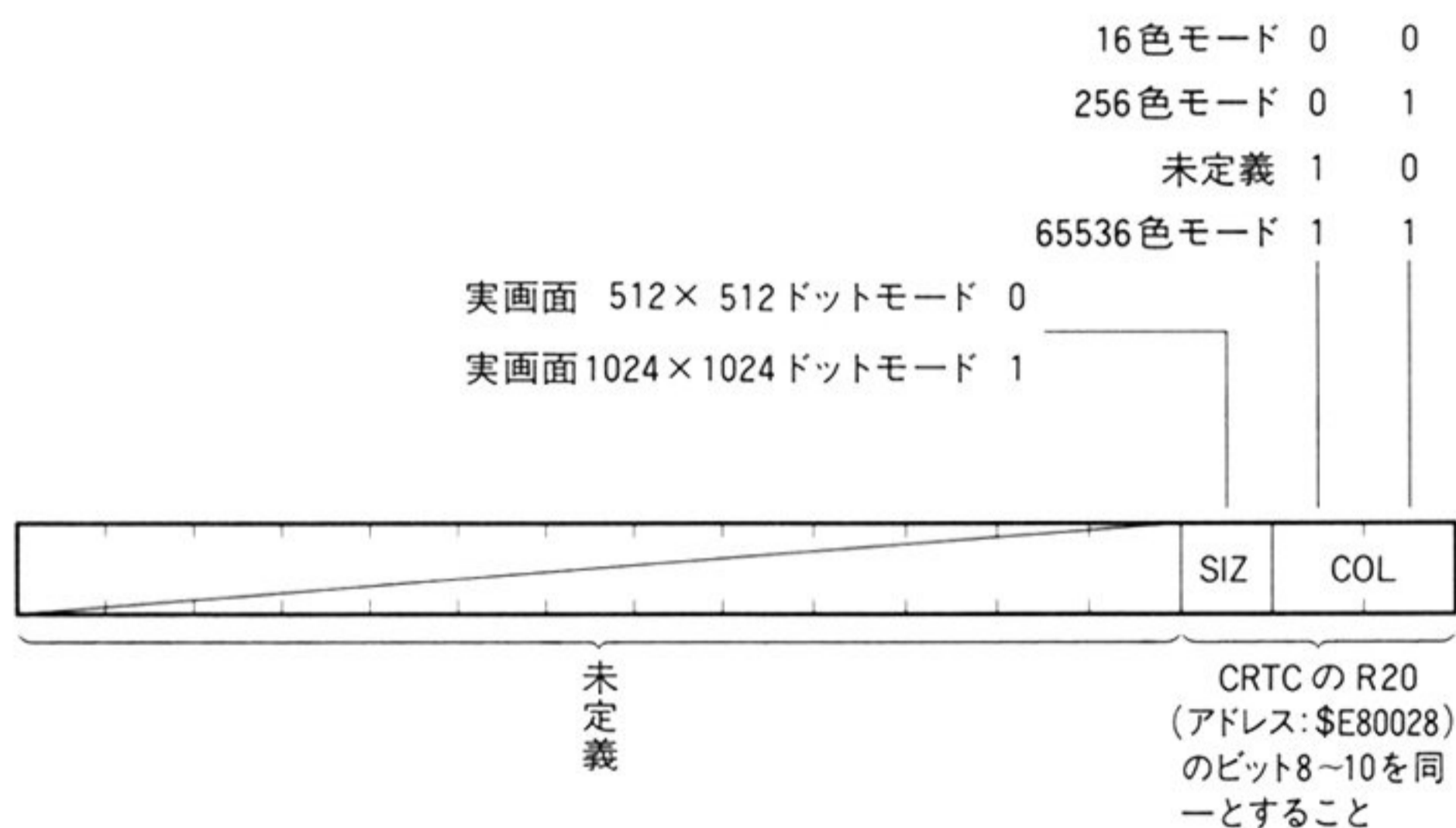
$$〔R05〕=\frac{(\text{垂直同期パルス幅})}{(\text{水平同期期間})}-1$$

$$〔R06〕=\frac{(\text{垂直同期パルス幅})+(\text{垂直バックポーチ})}{(\text{水平同期期間})}-1$$

$$〔R07〕=\frac{(\text{垂直同期期間})-(\text{垂直フロントポーチ})}{(\text{水平同期期間})}-1$$

●図……54　CRTC R20($E80028)

❺·2 ビデオコントローラ

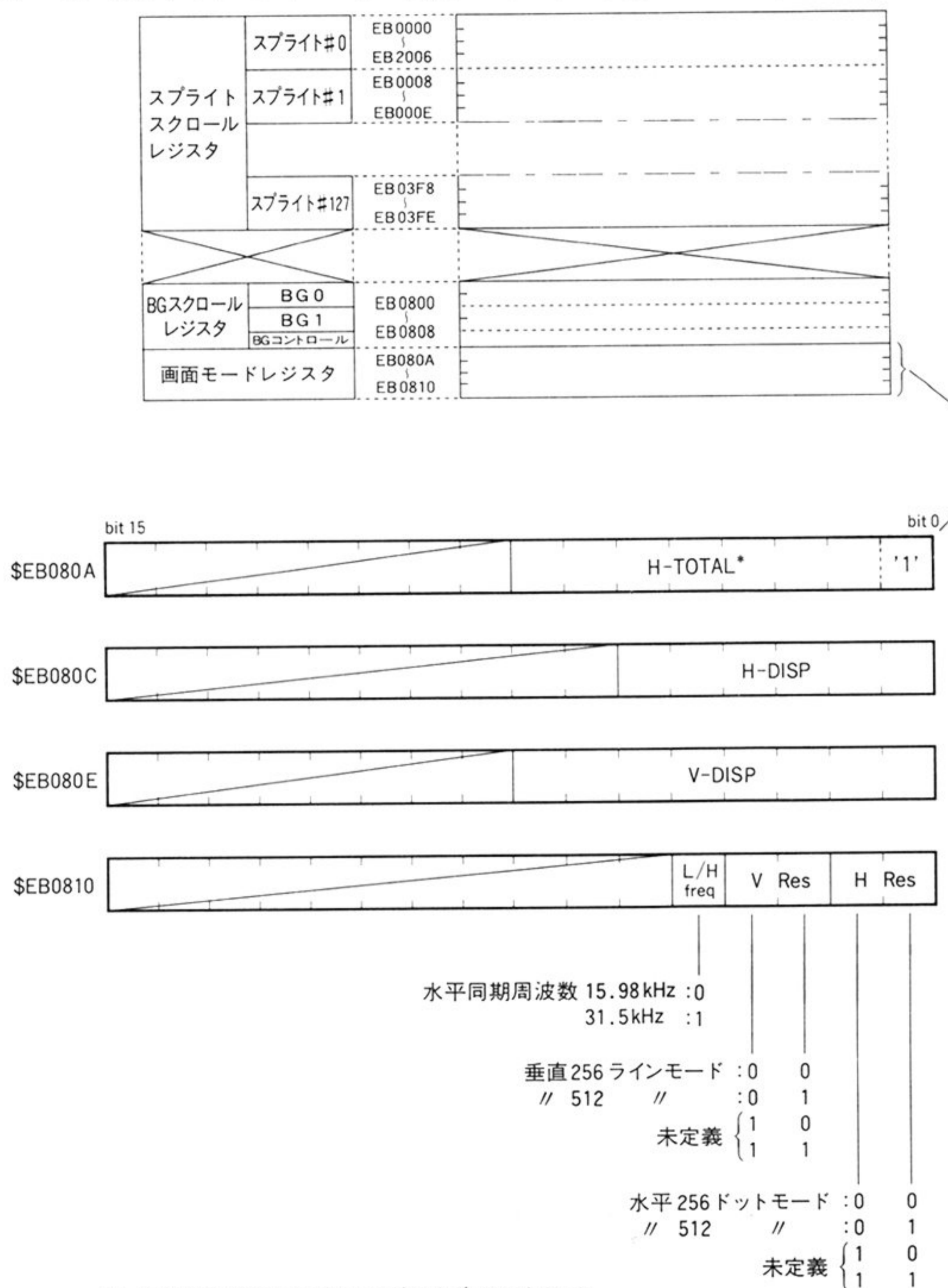
ビデオコントローラのレジスタのうち画面モードに関係するのはR0です。レジスタのビット配置を図55に示します。R0の下位3ビットは，画面の実画面モードや色モードの選択を行います。この設定値は，CRTCのR20のビット8，9，10の3ビットと同じ値になります。

●図……55　ビデオコントローラ R0($E82400)

❺·3 スプライトコントローラ

スプライトコントローラでは，画面モードに応じた設定を画面モードレジスタに行います。画面モードレジスタは4本の16ビット長のレジスタからなっており，それぞれのビット配置は図56のようになっています。これらのうち，H-TOTAL($EB080A番地)，H-DISP($EB080C番地)，V-DISP($EB080E番地)の設定の標準値は236ページの図57のようになっています。それぞれの設定値の計算法は次のようになっています。

1 H-TOTAL

低解像度の256×256ドットモードのときだけCRTCのR00と同じ値を，それ以外のとき

●図……56 スプライトコントローラ 画面モードレジスタ($EB080A～$EB080F)

には$FFを設定します。このレジスタへの設定値もR 00と同様，必ず奇数（最下位ビットがつねに'1'）にしてください。

●図……57 スプライトコントローラ　画面モードレジスタの標準設定値

画面モードレジスタ		高解像度モード			標準解像度モード		
名称	アドレス	512×512	512×256	256×256	512×512	512×256	256×256
H-TOTAL	$EB080A	$FF (255)	$FF (255)	$FF (255)	$FF (255)	$FF (255)	$25 (37)
H-DISP	$EB080C	$15 (21)	$15 (21)	$0A (10)	$09 (9)	$09 (9)	$04 (4)
V-DISP	$EB080E	$28 (40)	$28 (40)	$28 (40)	$10 (16)	$10 (16)	$10 (16)
	$EB0810	$15 (21)	$11 (17)	$10 (16)	$05 (5)	$01 (1)	$00 (0)
BG面の数		1	1	2	1	1	2

(　)内は10進数

2 H-DISP

CRTCのレジスタR 02の設定値に4を足した値を設定します。

3 V-DISP

CRTCのレジスタR 06と同じ値を設定します。

また，$EB0810番地のレジスタにはCRTCのレジスタR 20の下位8ビット（ビット0～7）と同じ値を設定してください。

5·4 設定上の注意

CRTCなどへの設定では，いくつか注意が必要な点がありますので，ここで補足しておきます。

5·4 1 CRTCへの設定時の注意

CRTCのレジスタR 00～R 07，R 20の設定を行う場合には，次のような順序で設定を行ってください。

・高い表示モードから低い表示モードに変更する場合

R 20→R 01→R 02→R 03→R 04→R 05→R 06→R 07→R 00

・低い表示モードから高い表示モードに変更する場合

R 00→R 01→R 02→R 03→R 04→R 05→R 06→R 07→R 20

画面モードの順序は，R 20のビット4，1，0の3ビットで判断され，高い順に並べると，次のような順序になります。

768×512 ドット	(高解像度モード)
512×512/512×256	(高解像度モード)
512×512/512×256	(標準解像度モード)
256×256	(標準解像度モード)

❺・❹ 2 画像取り込み時のCRTCへの設定

画像取り込み時には，CRTCのR 08を次の値に変更します。

・512×512/512×256ドットモード時 ……$9A
・256×256ドットモード時……………………$EB

❺・❹ 3 スプライト画面モードレジスタ設定時の注意

スプライトコントローラの画面モードレジスタのH-TOTALレジスタ（$EB080A番地）に$FF以外の値を設定するとき（標準解像度の256×256ドットモードにするとき）にはH-DISPレジスタの設定後，130 μs以上たってから行ってください。

❺・❹ 4 スプライトRAMアクセスの注意

電源投入後，スプライトVRAM（PCGエリア，BGデータエリア）のアクセス時は，BGコントロールレジスタ（$EB0808番地）のビット10を'0'に設定した後に行ってください。

●6 サンプルプログラム

CRTCやビデオコントローラの操作などを行うサンプルをいくつかつくってみましたので参考にしてください。

❻·1 テキスト画面スクロール(C1.C)

テキスト画面を上下，左右にスクロールします。わざと球面スクロールのように動かしています。はみ出したときの動作も見ておいてください。

●リスト……1　テキスト画面スクロール

```
/*
 *  テキスト画面スクロールサンプル
 * (横方向にはみ出してしまったときの動作も確認してください)
 *
 * XC ではvolatile がサポートされていないため、
 * 次の1行を入れてvolatileを無効にしてください
 * #define  volatile
 */
#include "doslib.h"
#define GP_VDISP    0x10
volatile short  *crtc_r10;
volatile short  *crtc_r11;
volatile char   *gpip;

void set_xpos(int xpos);
void set_ypos(int xpos);
void wait_vdisp(void);

main()
{
    int i,j;
    (short *)crtc_r10= 0xe80014;
    (short *)crtc_r11= 0xe80016;
```

```
    (char *)gpip      = 0xe88001;
    SUPER(0);
    for (i=0; i<1024; i+=4)
        set_ypos(i);
    for (i=0; i<1024; i+=4)
        set_xpos(i);
    for (i=1020; i>=0; i-=4)
        set_ypos(i);
    for (i=1020; i>=0; i-=4)
        set_xpos(i);
    exit(0);
}

void set_xpos(xpos)
    int xpos;
{
    wait_vdisp();
    *crtc_r10 = xpos;
}

void set_ypos(ypos)
    int ypos;
{
    wait_vdisp();
    *crtc_r11 = ypos;
}

void wait_vdisp()
{
    while(!(*gpip & GP_VDISP))
        ;
    while(*gpip & GP_VDISP)
        ;
```

❻・2 グラフィック画面4方向スクロール(C2.C)

512×512×16色×4プレーンのモードで，各画面を独立してスクロールさせています。

```
    apage(3);
    for (i=256; i<512; i++)
        line(i,256,i,511,i%16,'NASI');

    apage(0);
}

void wait_vdisp()
{
    while(!(*gpip & GP_VDISP))
        ;
    while(*gpip & GP_VDISP)
        ;
}
```

❻-3 ラスタコピー機能によるテキスト画面スクロール(C3.C)

ラスタコピー機能を使って，テキスト画面の上下スクロールを実現してみました。カーソル移動キーを操作すると画面が上下します。終了するときはCTRL+Cを入力してください。

●リスト……3　ラスタコピー機能によるテキスト画面スクロール

```
/*
 * ラスタコピー機能サンプル (テキスト画面スクロール)
 *
 * カーソル上下キーで、画面が上下し、CTRL+Cで終了します。
 *
 * XC ではvolatile がサポートされていないため、
 * 次の1行を入れてvolatileを無効にしてください
 * #define  volatile
 */
#include "basic0.h"
#include "doslib.h"

volatile short  *crtc_r21;
volatile short  *crtc_r22;
volatile short  *crtc_mode;
```

```
volatile char   *gpip;

char raster_scroll(void);
void raster_copy(int src, int dst);
void wait_h_sync(void);
void start_raster_copy(void);
void stop_raster_copy(void);

void main()
{
    int i;
    short   r21dat, r22dat;
    gpip      = (char *)0xe88001;
    crtc_r21 = (short *)0xe8002a;
    crtc_r22 = (short *)0xe8002c;
    crtc_mode = (short *)0xe80480;
    C_CUROFF();
    SUPER(0);
    r21dat = *crtc_r21;
    r22dat = *crtc_r22;

    while(raster_scroll() != 0x3)
        ;
    wait_h_sync();
    stop_raster_copy();
    *crtc_r21 = r21dat;
    *crtc_r22 = r22dat;
    C_CURON();
    exit(0);
}

char raster_scroll()
{
    char    keybuf[8];

    b_inkey0(keybuf);
    if (keybuf[0] == 0x1b) {
        b_inkey0(keybuf);
        switch(keybuf[0]) {
            case    0x55:   roll_up();
                    break;
            case    0x4a:   roll_down();
```

```
                    break;
            default:    break;
        }
    }
    return(keybuf[0]);
}

roll_up()
{
    int i;
    raster_copy(0,128);
    for (i=0; i<123; i++)
        raster_copy(i+1, i);
    raster_copy(128,123);
}

roll_down()
{
    int i;
    raster_copy(123,128);
    for (i=123; i>0; i--)
        raster_copy(i-1, i);
    raster_copy(128,0);

}

void raster_copy(src, dst)
    int src, dst;
{
    dst &= 0xff;
    src &= 0xff;
    wait_h_sync();
    stop_raster_copy();
    *crtc_r21 = 0x3;
    *crtc_r22 = (src << 8) | dst;
    start_raster_copy();
}

void start_raster_copy()

{
```

```
    *crtc_mode |= 0x8;
}


void stop_raster_copy()
{
    *crtc_mode &= 0x7;
}

void wait_h_sync()
{
    int dat;
    while((*gpip & 0x80) == 0x0)
        ;
    while((*gpip & 0x80) == 0x80)
        ;
}
```

❻·4 グラフィック画面の高速クリア(C4.C)

256×256ドットモードでグラフィック画面の高速クリア機能を使ってみました。カーソル移動キーで画面を動かしてクリアされているのが，実画面の一部だけであることを確認してください。

●リスト……4　グラフィック画面の高速クリア

```
/*
 *  グラフィック高速クリア機能サンプル
 *
 *  256×256ドットモード時での高速クリアです。
 *  カーソル移動キーで画面をスクロールさせて
 *  どのエリアがクリアされているか確認できます。
 *
 * XC ではvolatile がサポートされていないため、
 * 次の1行を入れてvolatileを無効にしてください
 * #define  volatile
 */
#include "basic0.h"
short   *vram;
```

```
volatile short  *crtc_r21;
volatile short  *crtc_mode;
volatile char   *gpip;
volatile short  *crtc_r12;
volatile short  *crtc_r13;


short   r21dat;
int pos_x,pos_y;

void init(void);
void h_clr(void);
void wait_v_sync(void);
char g_move(void);

main()
{
    screen(0,0,1,1);
    vram = (short *)0xc00000;
    gpip      = (char *)0xe88001;
    crtc_r21  = (short *)0xe8002a;
    crtc_mode = (short *)0xe80480;
    crtc_r12  = (short *)0xe80018;
    crtc_r13  = (short *)0xe8001a;
    pos_x = pos_y = 0;
    printf("High Speed Clear Test¥n");
    SUPER(0);
    r21dat = *crtc_r21;
    init();
    h_clr();
    while(g_move() != 3)
        ;
    *crtc_r21 = r21dat;
    exit(0);
}

void init()
{
    int i;
    short   col;
    for (i=0; i<1024*1024; i++)
        *vram++ = col++;
```

```
}

void h_clr()
{
    wait_v_sync();
    *crtc_r21 = 0xf;
    *crtc_mode = 0x2;
    wait_v_sync();
}

void wait_v_sync()
{
    while(*gpip & 0x40)
        ;
    while(!(*gpip & 0x40))
        ;
}

char g_move()
{
    char    keybuf[16];
    b_inkey0(keybuf);
    if (keybuf[0] == 0x8)
        *crtc_r12 = pos_x++;
    if (keybuf[0] == 0x1b) {
        b_inkey0(keybuf);
        switch(keybuf[0]) {
            case    0x53:   *crtc_r12 = pos_x--;
                    break;
            case    0x55:   *crtc_r13 = pos_y++;
                    break;
            case    0x4a:   *crtc_r13 = pos_y--;
                    break;
            default:    break;
        }
    }
    return(keybuf[0]);
}
```

❻·5 65536色モードでの 4 プレーン独立スクロール(C5.C)

グラフィック画面のスクロールレジスタを独立して制御してみると，65536 色モードで 4 プレーンが独立してスクロールできることがわかります。メーカで動作を保証している使い方ではありませんので，あくまでも参考ということにしてください。

●リスト……5　65536 色モードでの 4 プレーン独立スクロール

```
/*
 *  65536色モードでの4プレーン独立スクロール (参考)
 *
 * XC ではvolatile がサポートされていないため、
 * 次の1行を入れてvolatileを無効にしてください
 * #define  volatile
 */
#include    "basic0.h"
#include    "graph.h"

void main();
void init_screen();
void move_screen();
void delay();

volatile unsigned short *x0 = (unsigned short *)0xe80018;
volatile unsigned short *y0 = (unsigned short *)0xe8001a;
volatile unsigned short *x1 = (unsigned short *)0xe8001c;
volatile unsigned short *y1 = (unsigned short *)0xe8001e;
volatile unsigned short *x2 = (unsigned short *)0xe80020;
volatile unsigned short *y2 = (unsigned short *)0xe80022;
volatile unsigned short *x3 = (unsigned short *)0xe80024;
volatile unsigned short *y3 = (unsigned short *)0xe80026;


void main(argc,argv)
int argc;
char    *argv[];
{
    init_screen();
```

```
    SUPER(0);
    move_screen();
}

void init_screen()
{
    unsigned int    i,j,col;
    screen( 1,3,1,1);
    window(0,0,511,511);
    for (i=0; i<256;i++) {
        col = i*256;
        for (j=0; j<256; j++)
            pset(128+j,128+i,col+j);
    }
}

void move_screen()
{
    unsigned int    i;
    for (i=0; i<128; delay(),i++) {
        *x0 = 511-i;
        *y0 = 511-i;
        *x1 = 511-i;
        *y1 = i;
        *x2 = i;
        *y2 = 511-i;
        *x3 = i;
        *y3 = i;
    }
}

void delay()
{
    unsigned int    i;
    for (i=0; i<5000; i++)
        ;
}
```

❻·6 768×512ドットモードでの65536色表示(V1.C)

通常，768×512ドットモードではグラフィック画面は16色表示しかできないのですが，CRTCとビデオコントローラをだましてやると，768×512ドットの画面の中の512×512ドットの領域で65536色表示ができます。ドットの密度が高く，また縦横のドット間隔がほぼ等しくなるため，たとえば，512×512ドットモードでは縦100ドット，横100ドットの四角形が横長の長方形になってしまったのが，このモードでは正方形として表示されます。このモードも，メーカで動作を保証しているものではありませんのであくまでも参考としておいてください。

●リスト……6　768×512ドットモードでの65536色表示

```
/*
 *  ７６８×５１２ドットでの６５５３６色表示（参考）
 *
 * XC ではvolatile がサポートされていないため、
 * 次の１行を入れてvolatileを無効にしてください
 * #define  volatile
 */
#include "stdio.h"
#include "doslib.h"
main()
{
    short   *vram, *crtcr20, *vcr1, *palette;
    int i, h, s, v;
    vram    = (short *)0xc00000;    /* G-Vram Start Address */
    crtcr20 = (short *)0xe80028;    /* CRTC R20      */
    vcr1    = (short *)0xe82400;    /* Video Controller R1  */
    palette = (short *)0xe82000;    /* Palette Register */
    SUPER(0);
    screen(2,0,1,1);
    *crtcr20= 0x0316;
    *vcr1   = 3;
    for (i=0x0001; i<=0x10000; i+=0x0202) {
        *palette++ = i;
        *palette++ = i;
    }
    for (h = 0; h < 192; h++)
```

```
        for (v = 0; v < 32; v++)
            for (s = 0; s < 32; s++)
                *vram++ = hsv(h, s, v);
}
```

❻·7 グラフィック画面2面とテキスト画面の半透明動作(V2.C)

グラフィック2画面とテキスト画面の複合半透明動作です。なぜかX-BASICなどではサポートされていない半透明機能ですが，なかなかおもしろい効果が得られますので，もう少し見直してもよいのではないかと思います。

●リスト……7　グラフィック2画面とテキスト画面の複合半透明動作

```
/*
 *  グラフィック２面とテキスト画面の複合半透明機能サンプル
 *
 * XC ではvolatile がサポートされていないため、
 * 次の１行を入れてvolatileを無効にしてください
 * #define  volatile
 */
#include "basic0.h"
#include "graph.h"

#define GREEN   0xf800
#define RED 0x07c0
#define BLUE    0x003e
#define INTENS  0x0001

void init_palette();

unsigned short  *gpal = (unsigned short *)0xe82000;
volatile unsigned short *video_r1 = (unsigned short *)0xe82500;
volatile unsigned short *video_r2 = (unsigned short *)0xe82600;
main()
{
    screen(1,2,1,1);
    locate(20,10);
```

```
    printf("GR1 + GR2 + Text Half tone¥n");
    apage(0);
    fill(110,110,400,400,2);
    fill(128,128,384,384,3);
    apage(1);
    fill(100,100,255,255,7);
    SUPER(0);
    init_palette();
    *video_r1 = (*video_r1 & 0xff) | 0x2400;
    *video_r2 = (*video_r2 & 0xff) | 0x1f00;
}

void init_palette()
{
    int i;
    unsigned short  *p;
    p = gpal;
    for (i=0; i<0x100; i++)
        *p++ = 0;
    *gpal++ = 0;
    *gpal++ = BLUE;
    *gpal++ = RED;
    *gpal++ = BLUE | RED;
    *gpal++ = GREEN;
    *gpal++ = GREEN| BLUE;
    *gpal++ = GREEN| RED;
    *gpal++ = BLUE | RED | GREEN;

}
```

❻·8 BG画面設定＆スクロール(S1.C)

PCG登録とBG画面の設定，スクロールなどを行ってみました。

●リスト……8　BG画面設定＆スクロール

```
/*
 *   ＢＧ画面設定＆スクロールサンプル
 *
```

```
 * XC ではvolatile がサポートされていないため、
 * 次の1行を入れてvolatileを無効にしてください
 * #define  volatile
 */
#include "basic0.h"

volatile unsigned short *bgscrlx0= (unsigned short *)0x00eb0800;
volatile unsigned short *bgscrly0= (unsigned short *)0x00eb0802;
volatile unsigned short *bgctrl  = (unsigned short *)0x00eb0808;
volatile unsigned short *bgtext  = (unsigned short *)0x00ebc000;
volatile unsigned short *pcg     = (unsigned short *)0x00eb8000;
volatile unsigned short *spscrl  = (unsigned short *)0x00eb0006;
volatile unsigned short *videor3 = (unsigned short *)0x00e82600;
volatile unsigned short *videor2 = (unsigned short *)0x00e82500;
volatile unsigned short *palette = (unsigned short *)0x00e82220;

void main()
{
    unsigned int    i,j;
    screen(1,3,1,1);
    SUPER(0);
    *bgscrlx0 = 0;
    *bgscrly0 = 0;
    *videor3 |= 0x40;
    *videor2 = (*videor2 & 0xff) | 0x1200;
    for (i=0; i<0x10; i++)
    *palette++ = ((i & 1)?0x3e:0) | ((i & 2)?0x7c0:0) |
((i & 4)?0xf800:0) | ((i & 8)?1:0);
    for (i=0; i<0x80; spscrl += 4,i++)
        *spscrl = 0;
    for (i=0; i<0x10; i++)
        *pcg++ = 0x1111;
    for (i=0; i<0x10; i++)
        *pcg++ = 0x2222;
    for (i=0; i<0x10; i++)
        *pcg++ = 0x4444;
    for (i=0; i<0x10; i++)
        *pcg++ = 0x8888;
    for (j=0; j<4; j++)
        for (i=0; i<0x10; i++)
            *pcg++ = 0;
```

```
        *bgctrl = 0x201;
        for (i=0; i<0x800; i++)
            *bgtext++ = 0x0100;
        for (i=0; i<0x800; i++)
            *bgtext++ = 0x0101;
        for (i=0; i<1024; i++) {
            *bgscrlx0 = i;
            for (j=0; j<5000; j++)
                ;
        }
        for (i=0; i<1024; i++) {
            *bgscrly0 = i;
            for (j=0; j<5000; j++)
                ;
        }
        exit(0);
}
```

C O L U M N

CPU のアクセス可能な期間

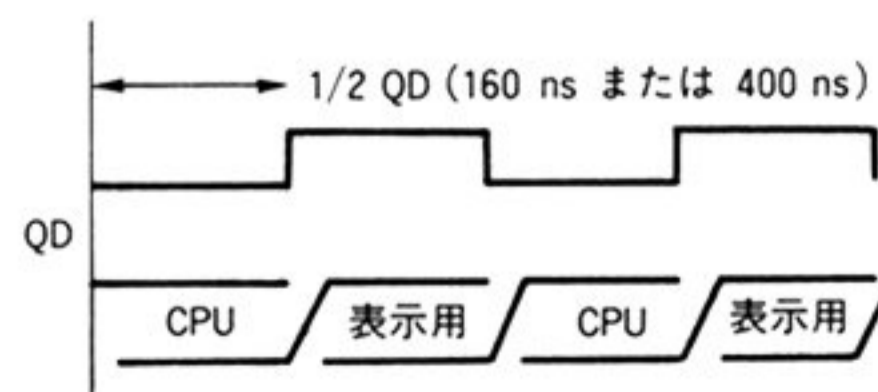
スプライトスクロールレジスタ

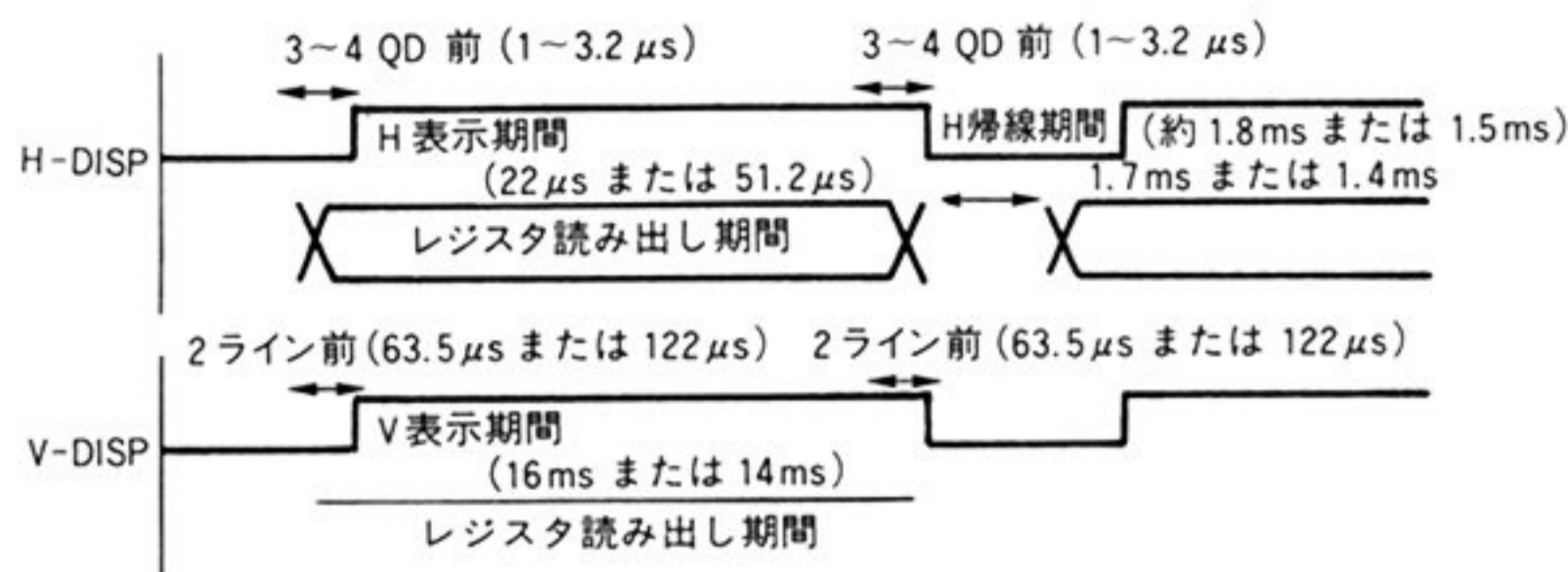
表示期間を含めたすべての期間においてアクセス可能です。

CPU 期間は，1キャラクタクロック（QD）に一度時分割で確保しています。そのため，最悪 1.8 QD（580 ns または 1440 ns）程度のウェイトがかかることがあります。

＊　QD の周期＝320 ns（高解像度）または 800 ns（標準解像度）

＊　DISP/CPU ビット（バックスクロールコントロールレジスタ；EB0808H の D09）を'0'に設定すれば，スプライトスクロールレジスタの時分割アクセスを禁止し，すべての期間を CPU に解放するため，高速なアクセスが可能になります。ただし，その期間，スプライト，バックグラウンドともすべての画面表示がカットされます。したがって，V 帰線期間に入ったら，まず，DISP/CPU ビットを'0'(表示カット）にした後，スプライトレジスタのアクセスを開始すれば能率的です。

●図……A　CPU アクセスのタイミング

表示用にレジスタを読み出す期間は以下のとおりです（表示時間と多少ずれがあります)。

●図……B　レジスタ読み出しのタイミング

したがって，V 帰線期間でスプライトスクロールレジスタを書き換える場合は，V-DISP の2ライン前までに書き換える必要があります。

なお，書き換えたスプライトスクロールレジスタの内容は，2ライン経過後，3ライン目で影響が現れます。

＊　標準解像度時，スーパーインポーズモードにした場合，H 帰線期間の一部で QD が停止することがあるため，CPU アクセスがその期間にぶつかると，ウェイトが延びることがあります（最悪 60 μs 程度)。

バックグラウンドスクロールレジスタ，および画面モードレジスタ

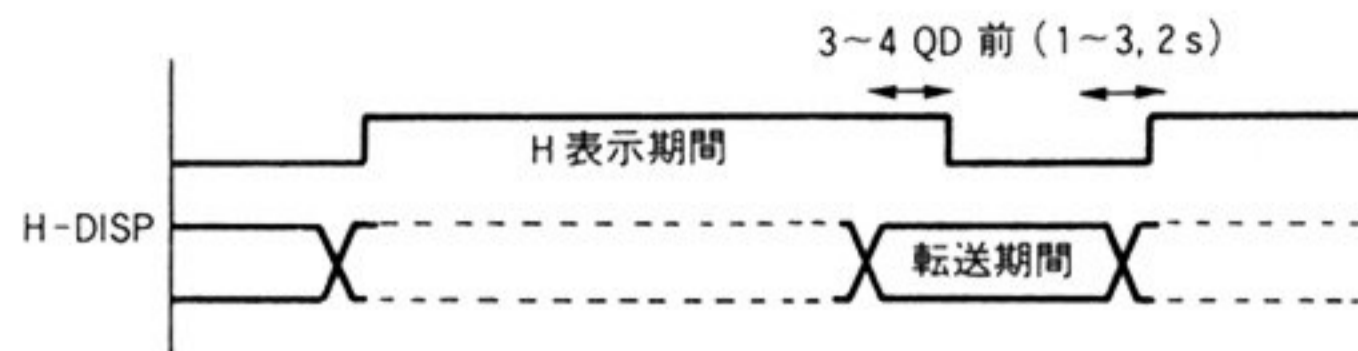
表示期間を含めたすべての期間においてアクセス可能です。

基本的に CPU 用レジスタと内部用レジスタの2段階のレジスタ構成です。CPU レジスタとは，CPU がアクセスできるレジスタで，通常1ウェイト以内でアクセスが終了します。CPU 用レジスタに書き込まれた内容は，1水平期間に一度，決まったタイミングで内部用レジスタへ転送され，チップ内部で有効になります。したがって，CPU がレジスタを書き換えたからといって，ただちに有効になるわけではありません。

＊　ただし，以下のビット情報については CPU 用レジスタのみで，2段階のレジスタ構成をとっていないため，CPU がレジスタを書き換えた直後からチップ内部で有効になります。
・バックグラウンドスクロールレジスタ……アドレス$EB 0808 DISP/CPU ビット（D 09)
・画面モードレジスタ……アドレス$EB 080 A H-TOTAL ビット（D 07～D 00)

CPU 用レジスタから内部用レジスタへ転送する期間は，ほぼ 256 ページの図Cのとおりです。

したがって，たとえば，バックグラウンドスクロールレジスタを1水平ラインごとに書き換えた場合は，1ライン前の水平表示期間中に書き換えればいいわけです（ただし，H-DISP の3～4 QD 前までに終了すること)。

●図……C　レジスタ転送のタイミング

転送期間と CPU の書き込みサイクルがぶつかった場合は，CPU 側にウェイトが入ります。読み出しサイクルならウェイトは入りません。

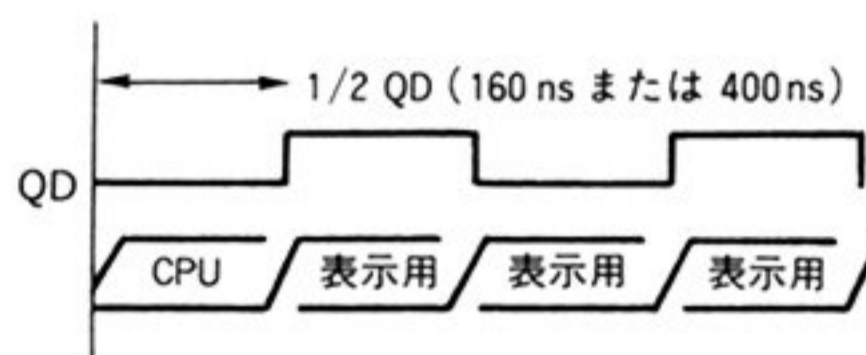
＊　バックグラウンドスクロールレジスタおよび画面モードレジスタへの CPU アクセスについては，スーパーインポーズの影響を受けません。

PCG およびテキスト

表示期間を含めたすべての期間においてアクセス可能です。

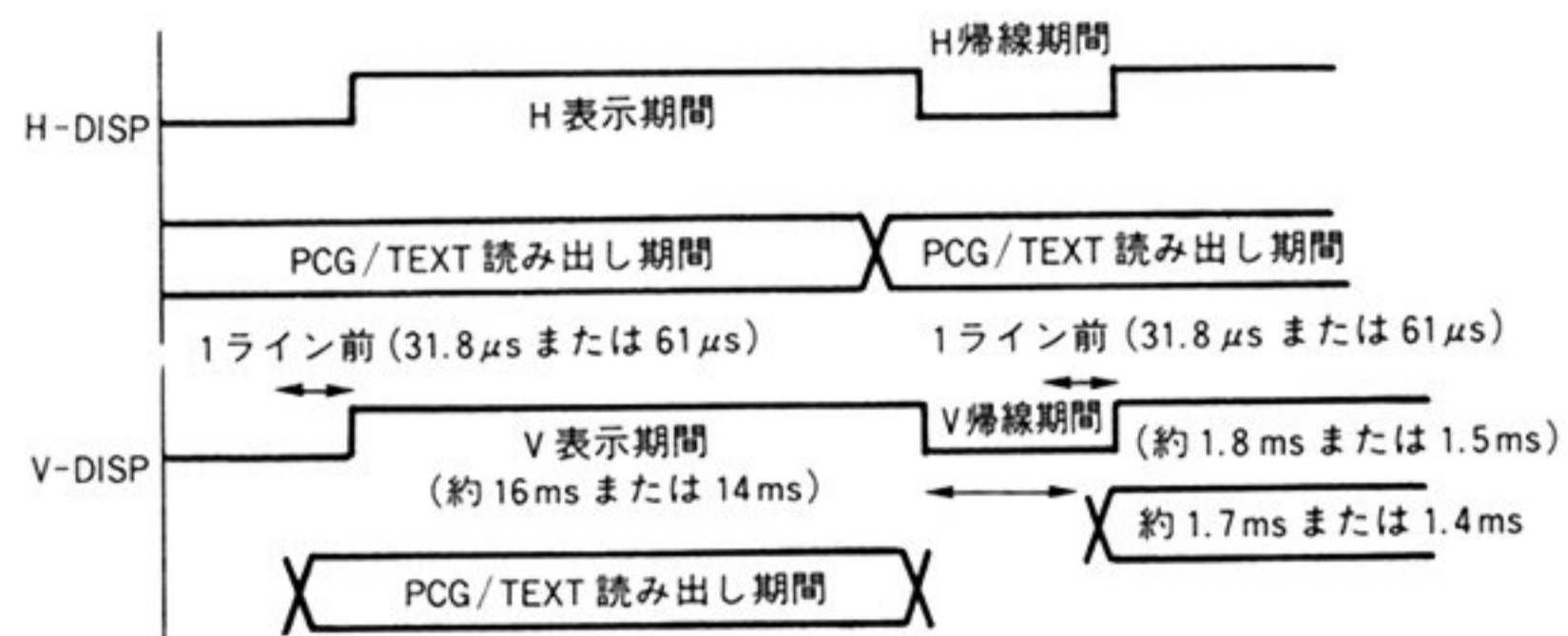
CPU 期間は，2 キャラクタクロック（QD）に一度，時分割で確保しています。そのため，最悪 2.8 QD（900 ns または 2240 ns）程度のウェイトがかかることになります。

＊　DISP/CPU ビット（バックグラウンドコントロールレジスタ；EB0808H の D 09）を'0'（CPU 側）に設定すれば，PCG，テキスト表示用の時分割が停止し，すべての期間を CPU に解放するため，高速なアクセスが可能になります。ただし，その間，スプライト，バックグラウンドともすべての画面表示がカットされます。したがって，V 帰線期間に入ったら，まず，DISP/CPU ビットを'0'（表示カット）にした後，PCG やテキストのアクセスを開始すれば能率的です。

●図……D　CPU アクセスのタイミング

表示用に PCG，テキストを読み出す期間は図 E のとおりです。

したがって，V 帰線期間内で PCG，テキストを書き換える場合は，V-DISP の 1 ライン前までに書き換える必要があります。また，H 帰線期間内では，書き換える期間はほとんどありません（表示モードによります）。

＊　標準解像度時，スーパーインポーズした場合，H 帰線期間内の一部で QD が停止することがあるため，CPU アクセスがその期間にぶつかると，ウェイトが延びることがあります（最悪 60 μs 程度）。

●図……E　レジスタ読み出しのタイミング

サウンド機構

FM音源とサンプリング(ADPCM)音源の両方を標準で搭載したX68000のサウンド機構は，効果音から音声出力までを幅広くサポートしています。ここでは，X68000のサウンド機構について説明します。

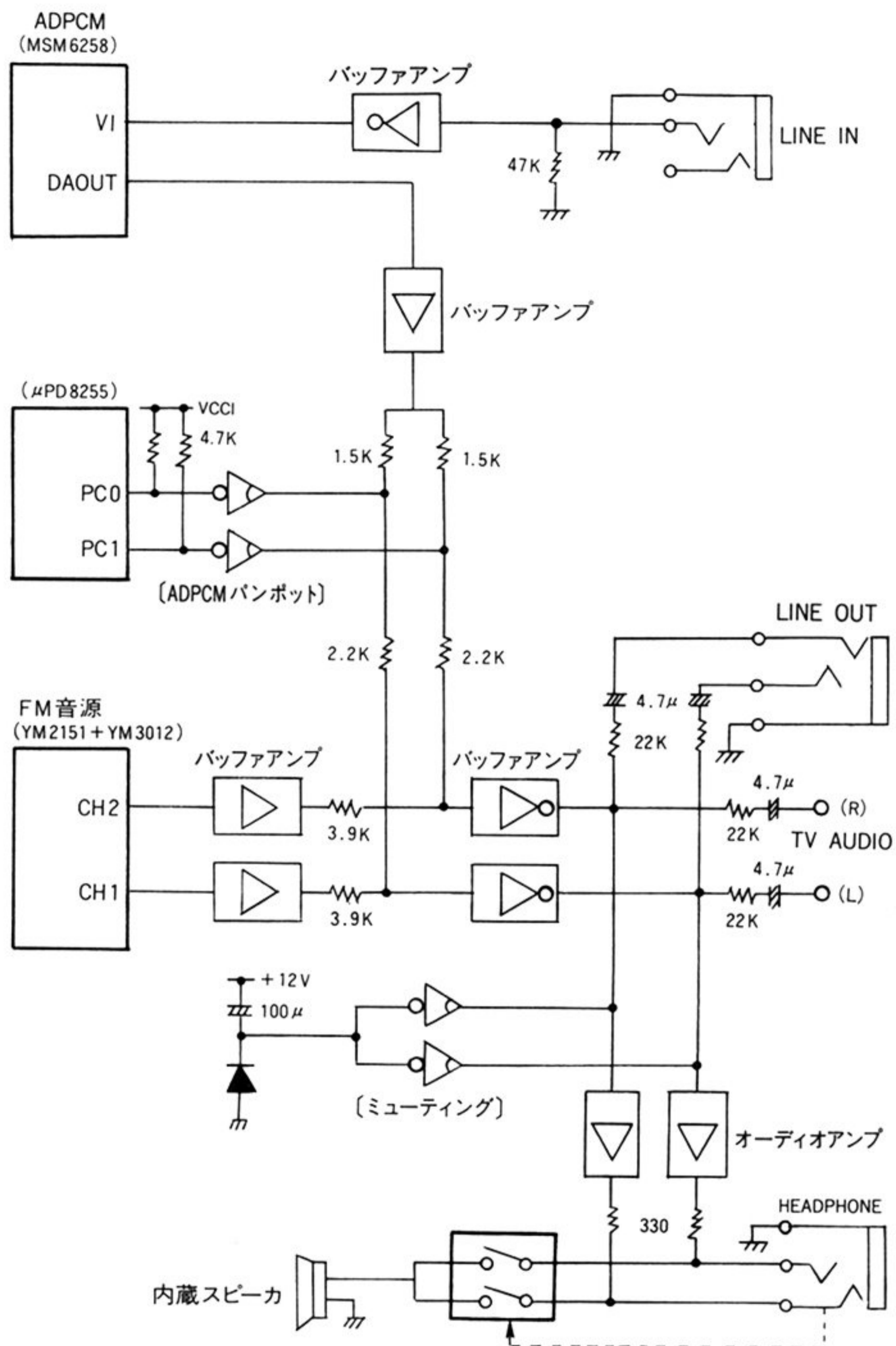
1 X68000のサウンド構成

X 68000のサウンド系統のブロック図を260ページの図1に示します。X 68000は，音声の取り込みや再生を行うADPCM音源と，正弦波を基本として純粋に演算処理で音を作成するFM音源の2つの音源用LSIを内蔵しています。

ADPCMの出力は，バッファアンプを通した後，左右に振り分けられて，FM音源ICの出力と合成されます。振り分けた後にある，μPD 8255の出力と接続されている部分はパンポット制御回路で，ADPCMの出力を右，左，中央のいずれに出力するか(あるいは，出力しないか)を決定する部分です。PC 0が左チャンネル，PC 1が右チャンネルに対応していて，出力が'1'(＝Highレベル)になっていると，該当するチャンネルへの出力がOFFにされます。両チャンネルともONになっていると，聴感上，中央から出力されているように聞こえます。

ハードウェアリセット後は8255のI/Oピンはすべて入力となるため，ADPCMの出力は両方ともOFFに，8255のイニシャライズ直後は出力ピンはすべて'0'(＝Lowレベル)となるため，ADPCM出力は両チャンネルともONになります。

オーディオアンプの前にも似たような回路が組んであります。これはどうやらミューティン

●図……1　サウンド系のブロック図

グ回路（電源投入時に「ボコッ」と大きな音が出てしまうのを防ぐ回路）のようです。

●2 FM音源

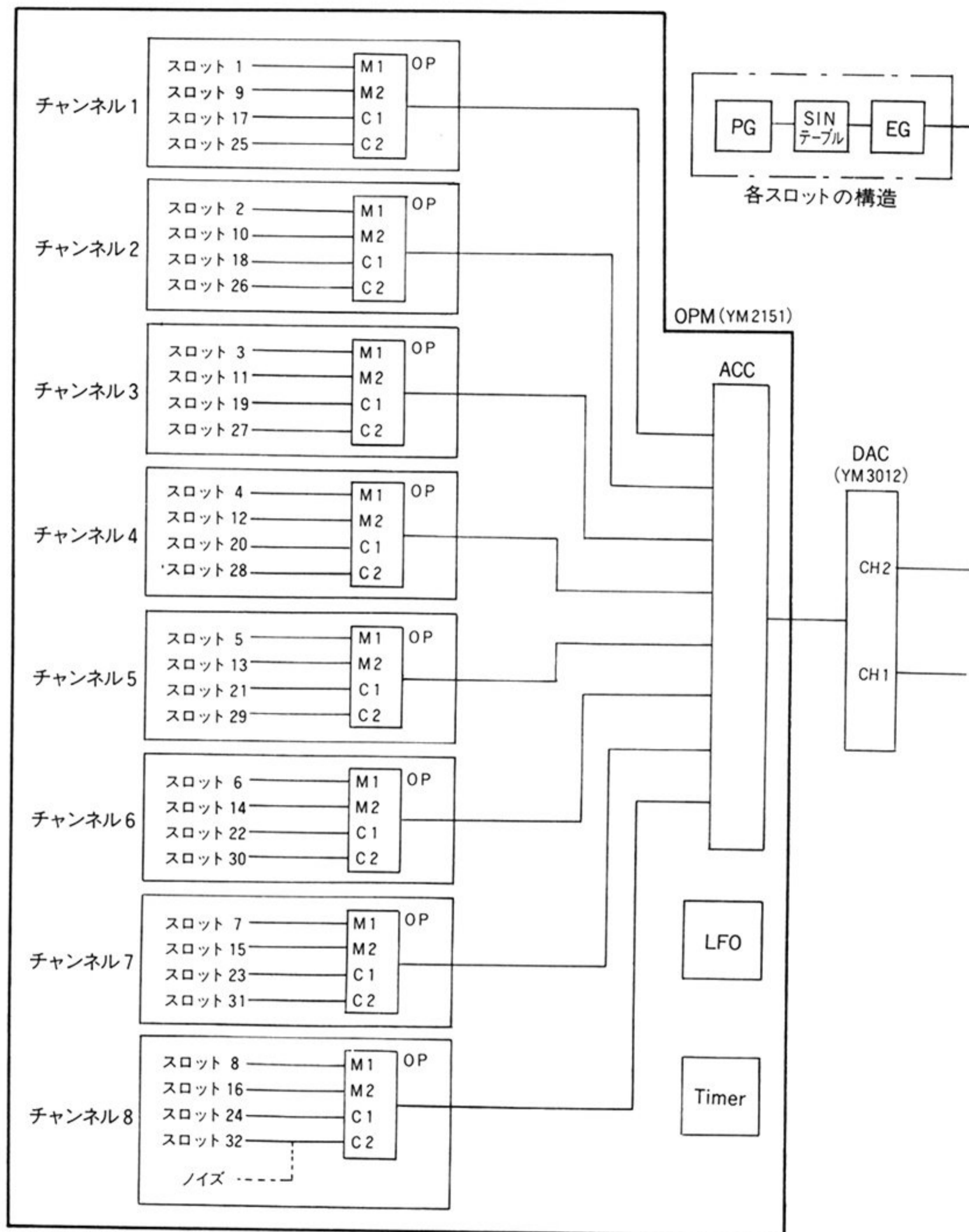
X 68000ではFM音源LSIとしてヤマハのYM 2151を使用しています。ヤマハでは，いくつもFM音源LSIを製造していますが，それぞれに愛称がついています。YM 2151はOPM (Fm Operator Type-m)という名称になっています。この名称はHuman 68 Kなどでも使用されているため，ご存じの方も多いでしょう。本書でもFM音源LSIの名称としてOPMを使用することにします。

❷·1 OPMの内部ブロック

OPMの内部ブロック図を262ページの図2に示します。OPMは8つの音声出力回路を持っており，それらの出力がミックスされて出力信号として取り出されます。それぞれのチャンネルは，チャンネル1からチャンネル8まで番号が振られており，通常の音楽演奏などでは，このチャンネル1つが楽器の1音に対応します。たとえば，ドミソの和音が必要なときはチャンネル1でド，2でミ，3でソの音を出力させるようにするわけです。

各チャンネルは4つのスロットと呼ばれる正弦波発振器からなっており，それぞれM1（モジュレータ1），M2（モジュレータ2），C1（キャリア1），C2（キャリア2）と名称がつけられています。これらを直列や並列につなぎあわせることで複雑な波形をつくり出すわけです。

OPMにはこのほか，音が出始めてから消えるまでのレベル変化(立ち上がりの強弱や余韻の長さなど）を制御するエンベロープジェネレータ（EGと略されます）や，ブルブルといった感じになる音の大きさの変化や，ワウワウといった周波数の変化（ビブラート）をつけるためのLFO(Low Frequency Oscillator)，あらかじめ設定した時間がくるとCPUに割り込みをかけたり，全部のスロットを一度に発声開始させることができるタイマ，「ザー」や「シー」といった音をつくるノイズ発生器（スロット32と切り替えて使用します）などが組み込まれています。

●図……2　OPM内部ブロック図

M1: Modulator 1　　OP　: FM Operator
M2:　〃　2　　LFO : Low Frequency Oscillator
C1 : Carrier 1　　ACC : Accumlator
C2 :　〃　2　　PG　: Phase Generator
EG　: Envelope Generator

❷・2 スロットの基本構造

OPMの発声単位であるスロットの基本構造と，それぞれを制御しているパラメータを264ページの図3に示します。

スロットの中心をなすのはSIN波形テーブルです。SIN波形テーブルは，角度データを与えると，それに対するsin値が出力として取り出されるテーブルです。このテーブルの入力として，0から2πまで直線的に変化し，次にふたたび0に戻るような，鋸波のデータを入力すれば，出力はきれいなサインカーブとなり，入力波形を歪めると，出力波形は大きく歪むことになります。OPMではスロットの入力に与える波形として，鋸波と他のスロットからの入力（M1スロットは自分自身の出力）を加算したものを与えることができるようになっています。たとえば，他のスロットからサイン波を与えると，SIN波形テーブルの出力Asinは，

$$\mathrm{Asin}=\mathrm{SIN}(\omega t+\alpha \mathrm{SIN}(\psi t))$$

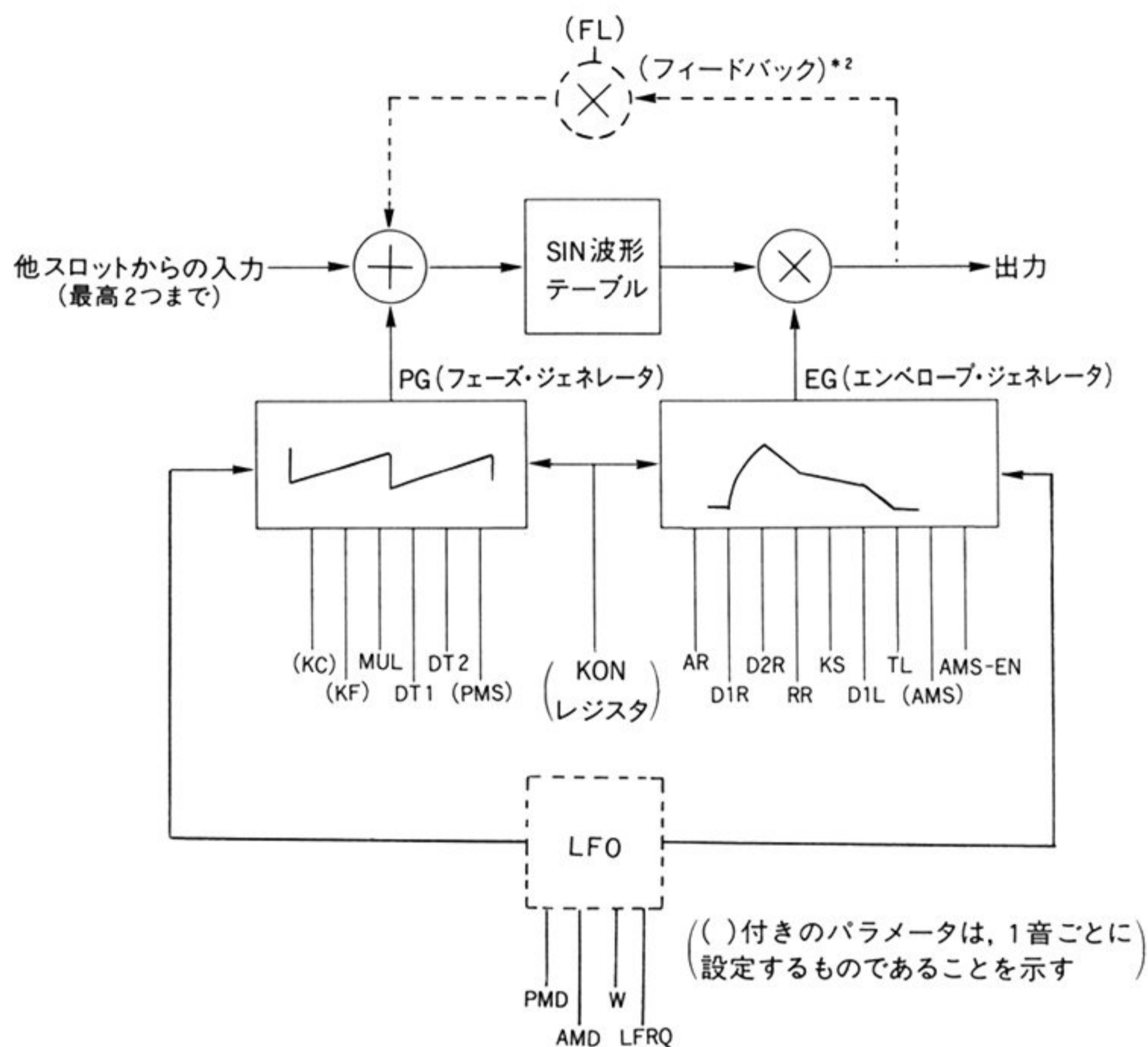
t　　：時刻

ψ，ω：角振動数

α　　：初段のスロットの出力レベルを決める値

となるわけです。先ほど触れた鋸波は，この式の中のωt，ψtの部分に相当します。OPMではこの信号を作成している部分をフェーズジェネレータ（PG）と呼んでいます。フェーズジェネレータを訳すと「位相生成器」となりますが，たんに「鋸波発振器」と考えておいてよいでしょう。フェーズジェネレータが発生する鋸波の基本周波数は，OPMのレジスタ中のKC，KF，MUL，DT1，DT2といったパラメータで決定され，さらにLFO（後述します）による変化の影響度をPMSで決定します。

さて，このSINテーブルの出力に実際に楽器を演奏したときに起こるような出力の時間的な変化を与えるのがエンベロープジェネレータ（EG）です。EGの波形は，アタック，ファーストディケイ，セカンドディケイ，リリースの4段階に分けられます（シンセサイザの世界などではファーストディケイの部分をたんにディケイ，セカンドディケイをサスティンと呼ぶのが一般的なようですが，ここではメーカのアプリケーションマニュアルに従った名称にしています）。これらを鍵盤楽器の場合にあてはめると，アタックはキーを押した直後の音の立ち上がり，ファーストディケイはアタックで行きついたところから少し戻るところ，セカンドディケイはキーを押し続けている間，音量が少しずつ下がっていくところ，リリースはキーから手を離した後の余韻に相当します。先ほどの式でいうと，初段のスロットの出力にかけ算されているαが初段スロットのEGの出力に相当します。当然，次段も独立したEGを持っていますか

●図……3　スロットの基本構造

＊1：フィードバックは，M1スロットだけにある

KC　:Key Code
KF　:Key Fraction
MUL:Phase MULtiply
DT1:De Tune 1
DT2:De Tune 2
PMS:Phase Modulation Sensitivity

AR　:Attack Rate
D1R:1'st Decay Rate
D2R:2'nd Decay Rate
RR　:Release Rate
KS　:Key Scaling
D1L:1st Decay Level
TL　:Total Level
AMS:Amplitude Modulation Sensitivity
AMS-EN:AMS Enable

PMD:Phase Modulation Depth
AMD:Amplitude Modulation Depth
W　:Waveform
LFRQ:Low FReQuency

ら，出力 Aout は EG の出力 β を使って，

$$\mathrm{Aout} = \beta \mathrm{SIN}(\omega t + \alpha \mathrm{SIN}(\psi t))$$

と表すことができます。

EG の出力波形は，OPM のレジスタ中の AR，D1R，D2R，RR，KS，D1L，TL といったパラメータで決定され，さらに LFO による出力レベル変動の ON/OFF や変動の度合を AMS-EN や AMS で決定しています。

図 3 の PG，EG のパラメータのうち，（　）でくくったものは 1 チャンネルごとに設定するものであることを，くくっていないものは 1 スロットごとに設定するものであることを示します。

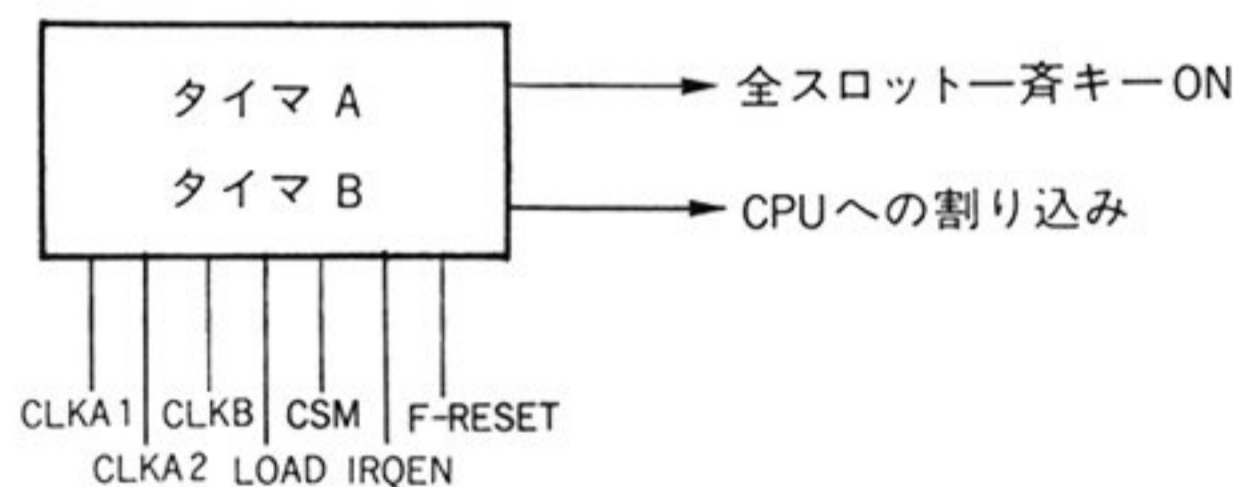
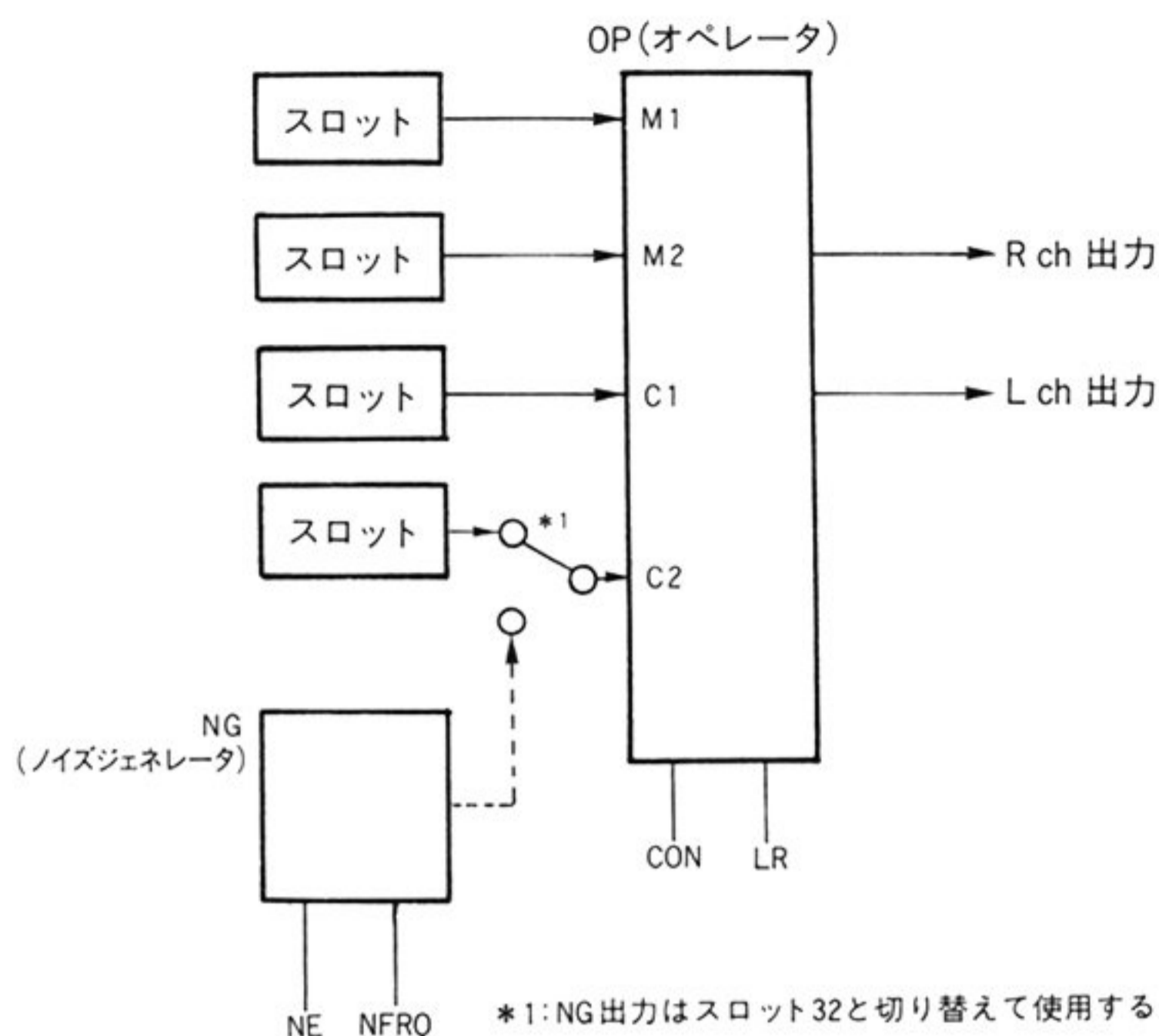
図 3 の，点線で示された LFO というブロックは，PG や EG の出力を低い周波数でふらつかせるための信号発生器です。OPM ではチップ内に 1 つだけ持っており，この出力をすべてのスロットが共通で使用しています。LFO は PG 用と EG 用の 2 つの出力を持っており，それぞれの出力レベルを PMD，AMD というパラメータで指定します。LFO の出力波形の種類，周波数はそれぞれ W，LFRQ というパラメータで決定されます。

M 1 スロットだけは自分自身の出力を自分の入力信号とするフィードバック回路を持っており，このフィードバック量を FL パラメータで指定するようになっています

❷·3　その他の部分の基本構造

OPM のスロット以外の部分の基本構造を 266 ページの図 4 に示します。オペレータには，スロットの組み合わせを指定する CON，チャンネルの音を左右，中央のいずれから出力するかを指定する LR パラメータが入力されます（メーカが公表している OPM のブロック図では，LR はアキュームレータの部分に入力されているのですが，感覚的にはオペレータに効いていると考えるほうがわかりやすいので，ここではオペレータに入力されるものとしています)。

ノイズジェネレータは OPM 内に 1 つだけあります。ノイズ出力の ON/OFF は NE，音質を NFRQ で指定します。ノイズ出力が ON されると，スロット 32 の SIN 波形テーブルの出力がノイズジェネレータの出力と置き換えられます（図では表しにくいので，スロットの出力と切り替えるように書いています)。

●図……4　スロット以外の部分の基本構造

NE：Noise Enable
NFRQ：Noise FReQuency
CON：CONnection
LR：Left channel Enable/
Right channel Enable

❷·4　OPMのアドレス配置

OPMのポートアドレスを図5に示します。

CPUは，$E90001番地にレジスタ番号を設定した後，$E90003番地のデータポートを使って，これらのレジスタにアクセスします。OPMは内部に多くのレジスタを持っていますが，音作りに関係するレジスタはすべて書き込み専用であり，リード時はレジスタ番号の設定に関係なく，つねにステータスレジスタが読み出されます。

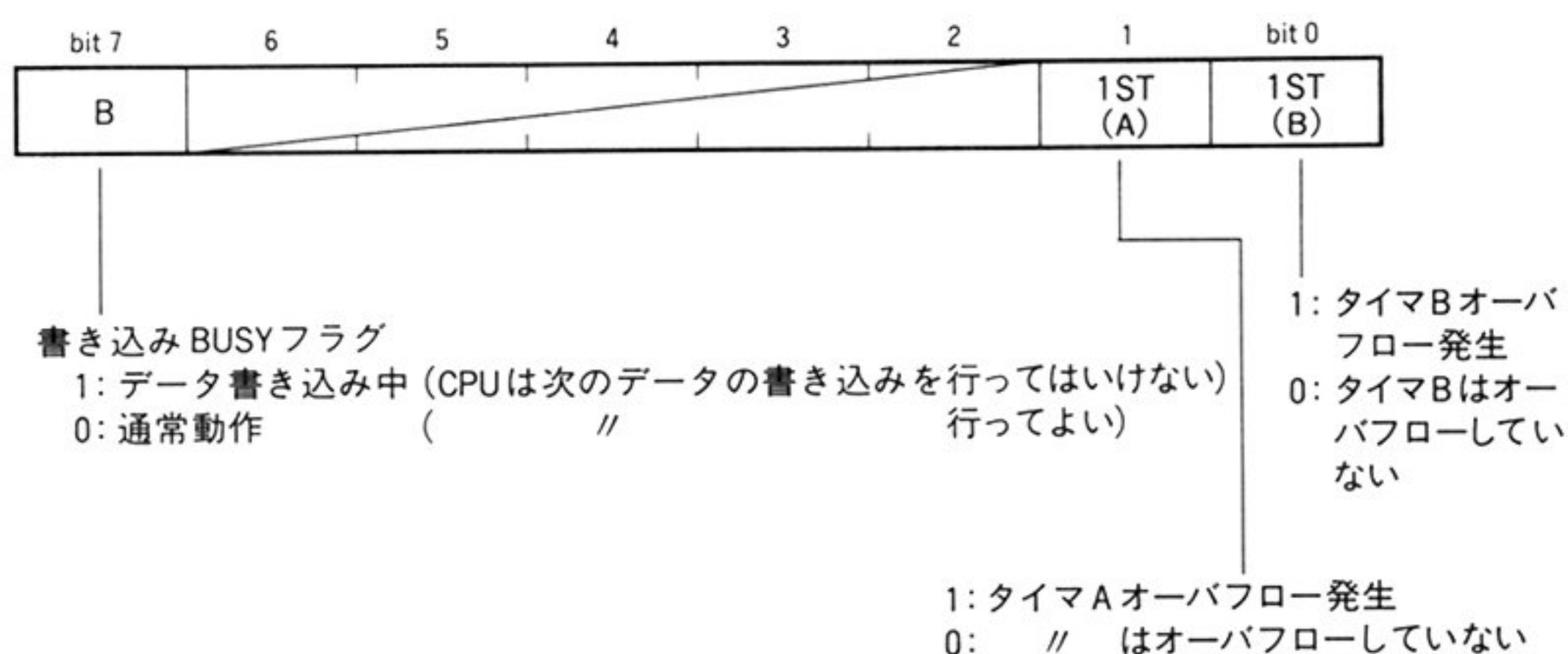
●図……5　OPM のポートアドレス

アドレス	データ (D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0)	内　容
$E 90001		レジスタ番号設定ポート
$E 90003		データRead/Writeポート

❷·5 OPMのリードレジスタ

OPM からのリードを行うと，つねにステータスレジスタの内容が読み出されます。このレジスタのビット配置を図6に示します。ビット7はOPMの書き込み BUSY フラグで，OPM が CPU から次のアクセスを受け付けられない状態であることを示しています。OPM への書き込みを行う場合は，このビットが '0' になっていることを確認してから行わなければなりません。

ビット0とビット1は，OPM 内部の2つのタイマのうち，いずれがオーバーフローしたのかを示すビットです。OPM のタイマは，あらかじめ設定した時間が経過した後に CPU に割り込みをかけたり，全スロットに一斉にキーオンを与えることができます。このビットはおもに CPU に割り込みをかけるような使い方をしたとき，割り込みがいずれのタイマによるものであるかを CPU が判断できるようにするために使用します。

●図……6　OPM ステータスレジスタ

❷·6 OPMのライトレジスタ

OPMの書き込みレジスタの配置を図7と図8に示します。レジスタ番号が$00から$1Fまでのレジスタはノイズ，タイマ，LFOなど，OPM内部に1つしかないものの設定，$20から$3Fがチャンネル単位で指定するもの，$40から$FFはスロット単位で指定するものが配置されています。次に，これらのレジスタをアドレス順に説明していくことにしましょう。

●図……7　OPMのレジスタ一覧（その1）

レジスタ番号	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	備考
$00									
$01	'0'						LFO Reset	'0'	テスト用レジスタと兼用なので，bit1以外のビットを'1'にしないこと
$02 〜 $07									
$08		KON (C2)	(C1)	(M2)	(M1)	(CH No)			Key ON/OFF 制御
$09 〜 $0E									
$0F	NE			NFRQ					$\text{fnoise(Hz)} = \frac{4000}{32 \times \text{NFRQ}}$ （ノイズ制御）
$10	CLKA（上位）								$T_A\text{(ms)} = \frac{64 \times (1024 - \text{CLKA})}{4000}$ （タイマA）
$11							CLKA（下位）		
$12	CLKB								$T_B\text{(ms)} = \frac{1024 \times (256 - \text{CLKB})}{4000}$ （タイマB）
$13									
$14	CSM		F-RESET (B)	(A)	IRQEN (B)	(A)	LOAD (B)	(A)	タイマ動作制御
$15 〜 $17									
$18	LFRQ								LFO周波数設定
$19	F	PMD/AMD							F=1: PMD =0: AMD
$1A									
$1B	CT1	CT2					W		汎用出力制御（CT1/CT2） LFO出力波形選択（W）
$1C 〜 $1F									

●図……8　OPMのレジスター覧（その2）

レジスタ番号	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	備　考	
$20〜$27	R-ch EN	L-ch EN	FL			CON			R/L 出力イネーブル（チャンネルの出力ON/OFF） FL：Feedback Level（M1のフィードバック量制御） CON：Connection（スロットの結合方式選択）	1音ごとに設定
$28〜$2F		KC (OCT)			KC (NOTE)				音階選択	
$30〜$37	KF								音階微調整	
$38〜$3F		PMS					AMS		PMS：Phase Modulation Sensitivity AMS：Amplitude Modulation Sensitivity	
$40〜$5F		DT1			MUL				DT1：Detune I（周波数微小変化） MUL：Phase Multiply（周波数倍率設定）	1スロットごとに設定
$60〜$7F		TL							TL：Total Level（出力レベル設定）	
$80〜$9F	KS			AR					KS：Key Scaling（EGの各レートのKCへの依存度選択） AR：Attack Rate	
$A0〜$BF	AMS EN			D1R					AMS EN：AMS Enable D1R：1'st Decay Rate	
$C0〜$DF	DT2			D2R					DT2：Detune 2（周波数大変化） D2R：2'nd Decay Rate	
$E0〜$FF	D1L				RR				D1L：1'st Decay Level RR：Release Rate	

❷・❻1　テストレジスタ

テストレジスタのビット配置を図9に示します。

●図……9　テストレジスタ

このレジスタは，メーカの出荷検査時に使うのがおもな目的です。公開されているのはビット1だけで，これ以外のビットはすべて'0'で使うようにしてください。ビット1を'1'にするとLFOがリセットされ，'0'に戻すとLFOがスタートします。他の音とLFOを同期させて動かしたいような場合に有効です。

❷·❻2 KONレジスタ

キーオン/キーオフの制御を行うレジスタです。ビット配置を図10に示します。音のON/OFFを制御するもので，鍵盤楽器でいうと，鍵盤を押したときがキーオン，鍵盤から指を離したときがキーオフとなります。

KONレジスタの下位3ビットで制御したいチャンネルを，ビット3～6で，そのチャンネルの各スロットをキーオンするか，キーオフするかを決めます。該当するビットが '1' のときにキーオン，'0' のときにキーオフになります。

KONレジスタでは1チャンネルずつしかONできないため，複数のチャンネルをキーオン

●図……10 KON

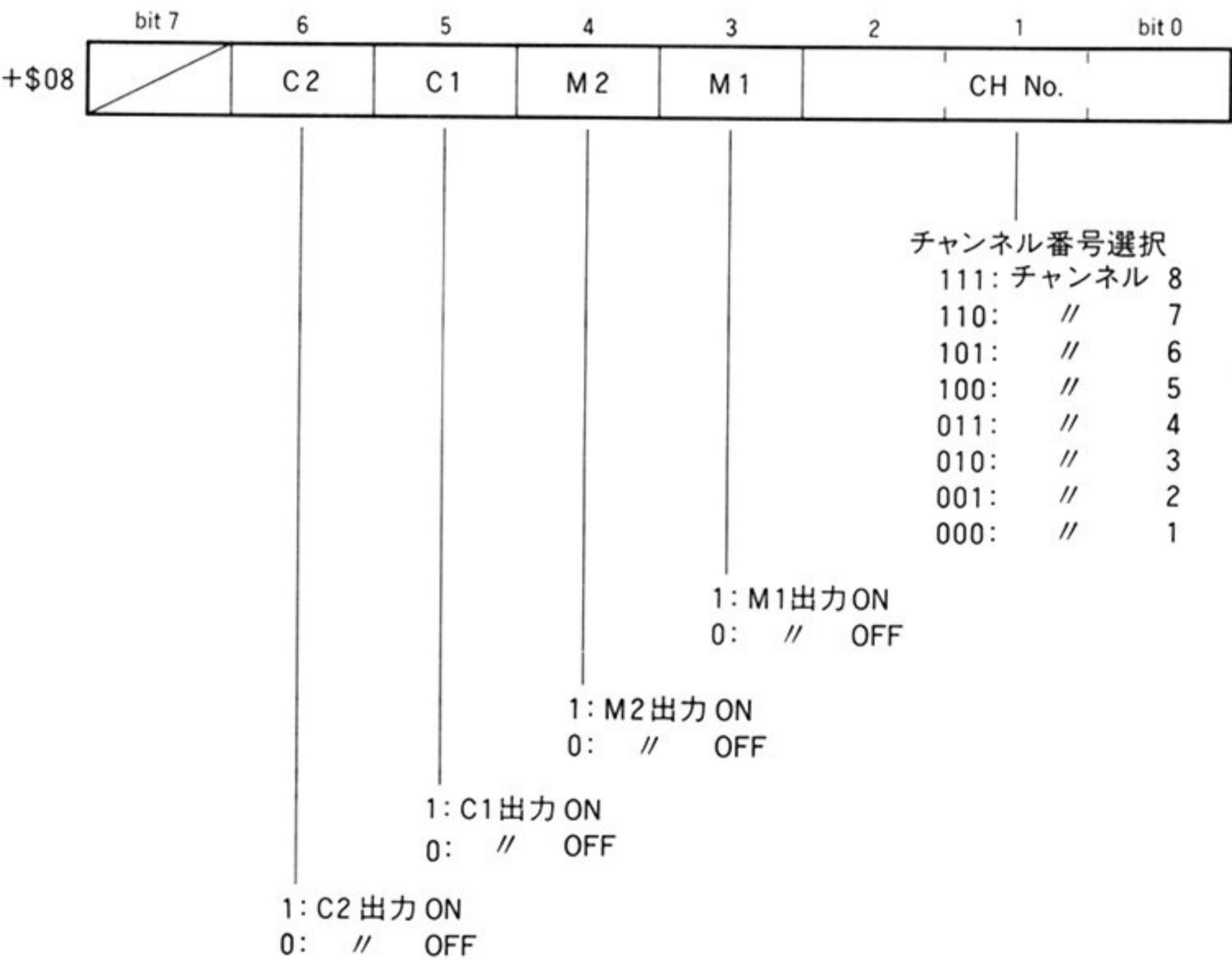
	bit 7	6	5	4	3	2	1	bit 0
+$08		C2	C1	M2	M1		CH No.	

チャンネル番号選択
111: チャンネル 8
110: 〃 7
101: 〃 6
100: 〃 5
011: 〃 4
010: 〃 3
001: 〃 2
000: 〃 1

1: M1出力ON
0: 〃 OFF

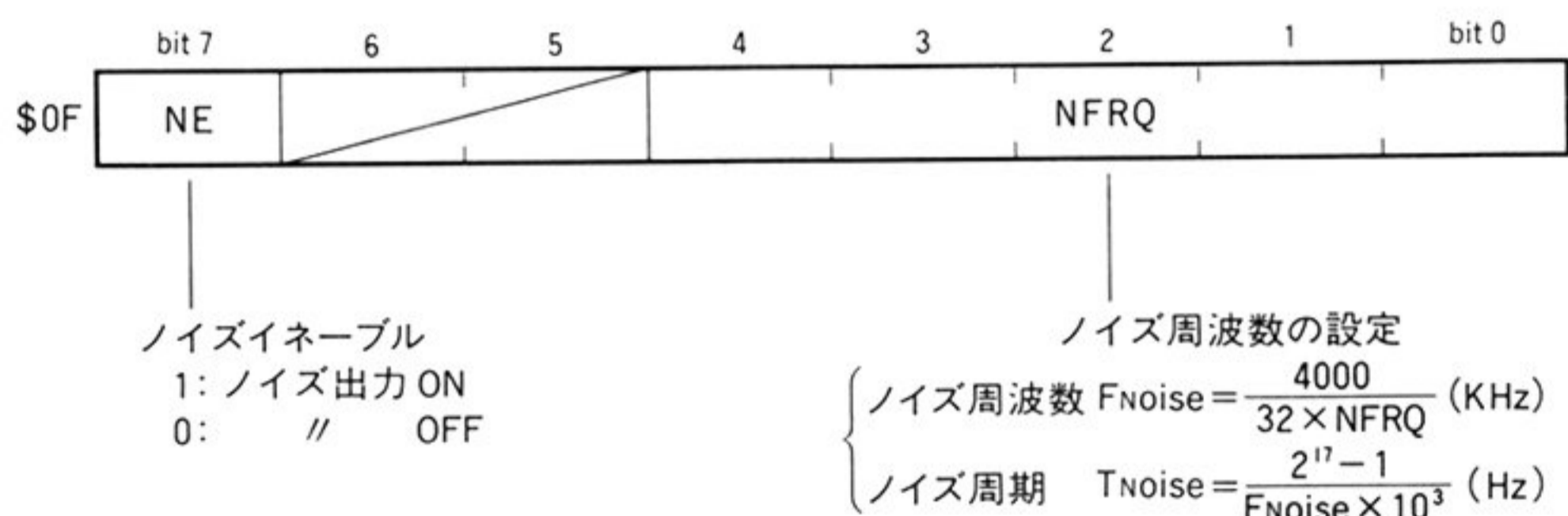
1: M2出力ON
0: 〃 OFF

1: C1出力ON
0: 〃 OFF

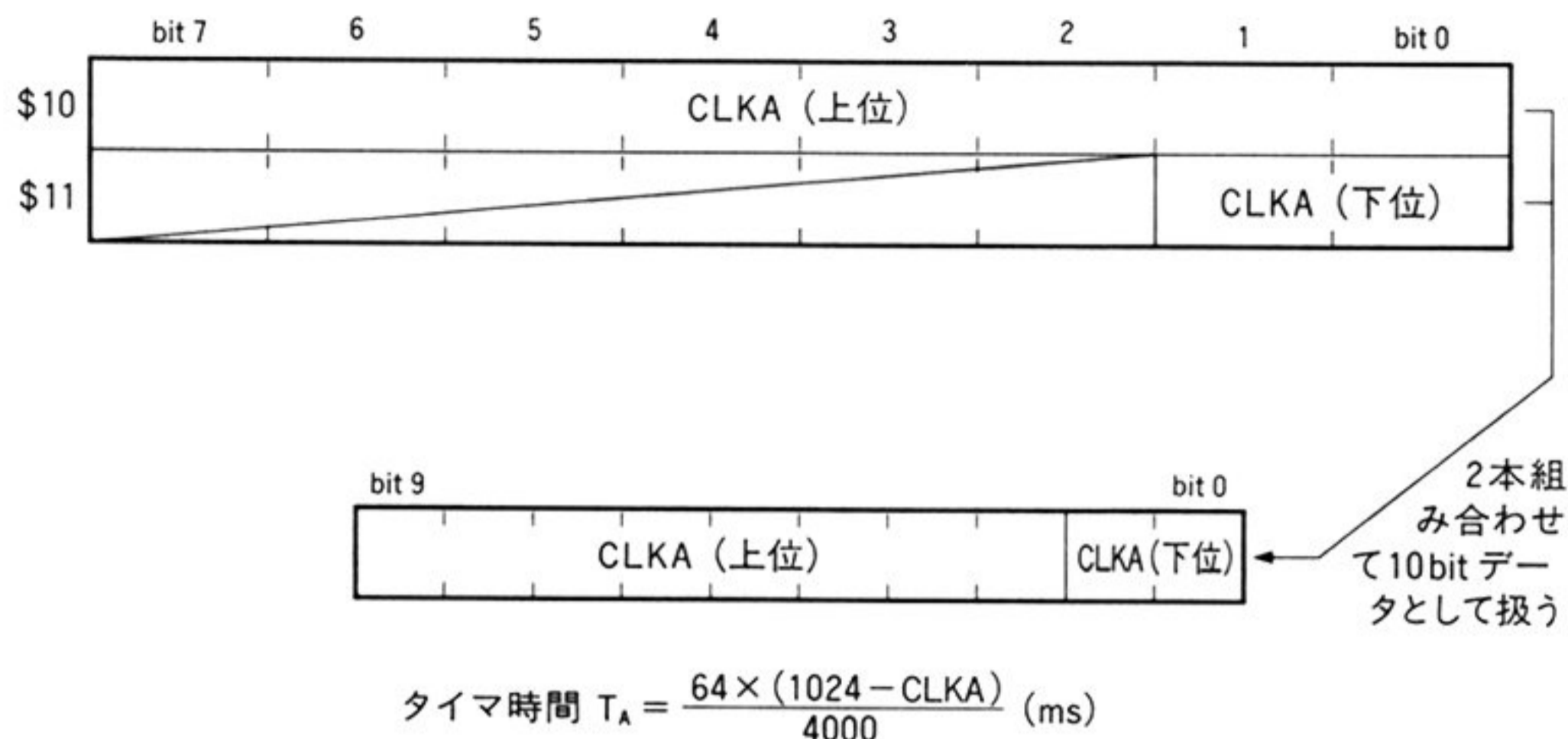
1: C2出力ON
0: 〃 OFF

させたときは，当然のことながら，それぞれのチャンネルの出力波形の位相はあわなくなります。位相を確実にあわせてスタートさせたいときはタイマAによるキーオン機能を利用します。

❷・❻ 3 ノイズジェネレータ制御レジスタ

OPM内部のノイズジェネレータのON/OFF制御やノイズ周波数の制御を行うレジスタです。ビット配置を図11に示します。下位5ビットでノイズ周波数を，ビット7でノイズジェネレータのON/OFF制御を行います。ビット7が'1'だとノイズジェネレータがイネーブルになり，スロット32がノイズと入れ替わります。エンベロープジェネレータはスロット32のものがそのまま使用されますが，エンベロープのカーブはアタックがエクスポーネンシャル，それ以外はすべて直線的に変化するようになります。

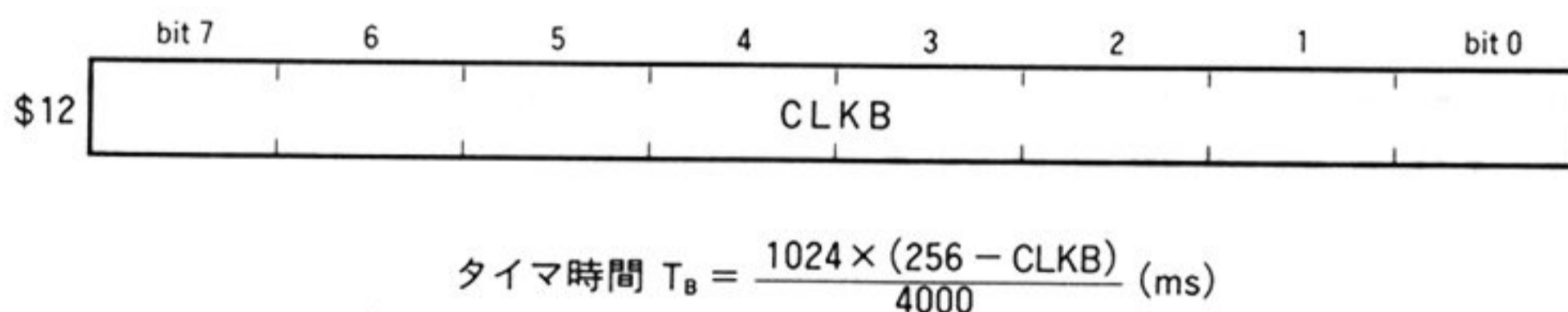
●図……11　ノイズジェネレータ制御

❷・❻ 4 タイマA設定レジスタ

OPMが持っている2本のタイマのうち，タイマAに時間設定を行うレジスタです。ビット配置は272ページの図12のようになっています。データ長が10ビットあるので，上位8ビットをレジスタ番号$10に，下位2ビットを$11に割り付けています。このレジスタに設定した値をCLKAとすると，64×(1024−CLKA)/4000 (ms)後にオーバフローを起こします。CPUに割り込みをかけたり，全スロットに一斉にキーオンを与えることができます。

●図……12　タイマA設定

❷·❻5　タイマB設定レジスタ

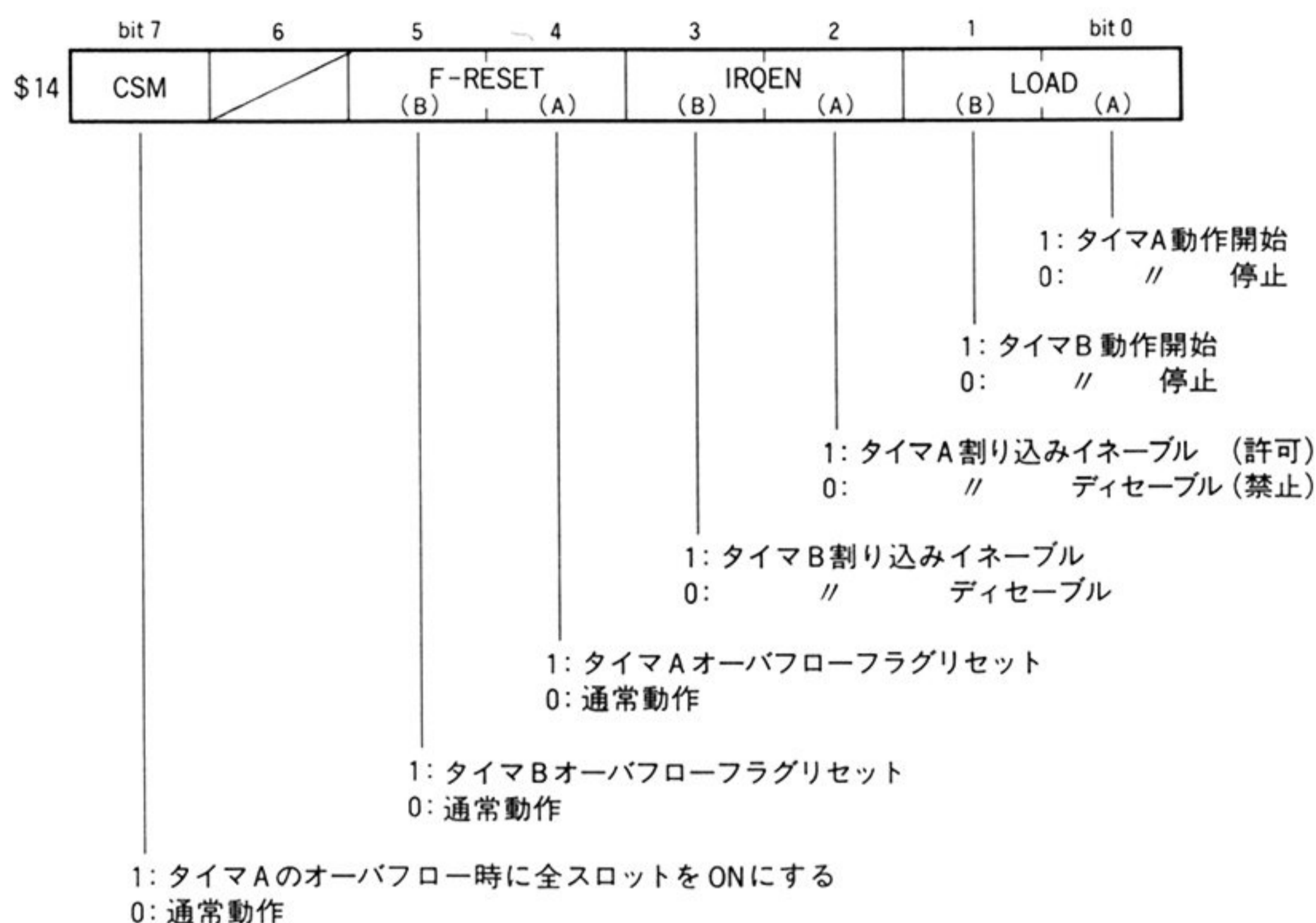
タイマBに時間設定を行うレジスタです。ビット配置は図13のようになっています。タイマBはビット長が8ビットであるため，レジスタ番号$12だけでカウント値が設定できます。このレジスタに設定する値をCLKBとすると，1024×(256−CLKB)/4000 (ms)後にCPUに割り込みをかけることができます。

●図……13　タイマB設定

$$タイマ時間\ T_B = \frac{1024 \times (256 - CLKB)}{4000}\ (ms)$$

❷·❻6　タイマ制御レジスタ

タイマ制御レジスタのビット配置を図14に示します。このレジスタは，タイマの動作ON/OFF制御や割り込みを発生するか否かなどを指定するものです。

ビット0，1はタイマ動作のON/OFFを制御するもので，ビット0がタイマAに，ビット1がタイマBに該当し，'1'でタイマが始動，'0'で停止します。

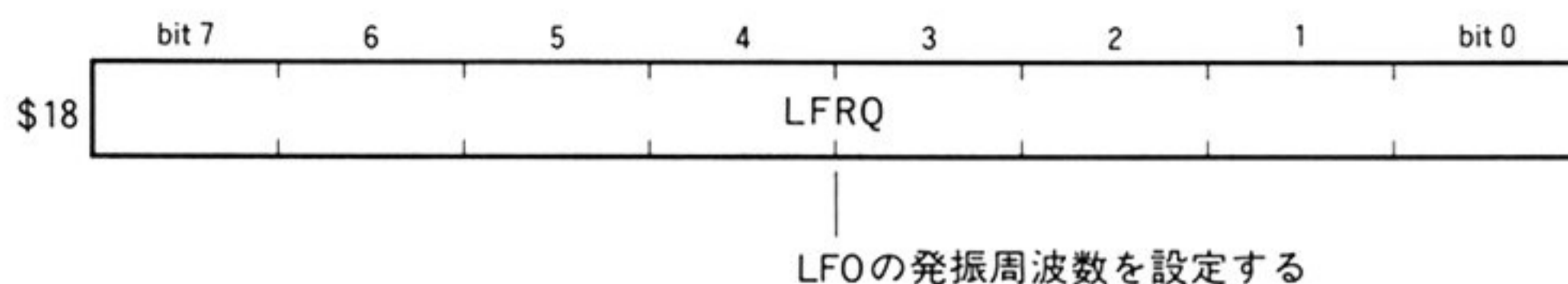
●図……14　タイマ制御

ビット2，3は，タイマがオーバフローしたときにCPUに割り込みをかけるか否かを設定するもので，ビット2がタイマA用，ビット3がタイマB用です。このビットを'1'にしておくと，オーバフロー発生時にCPUに割り込み発生を許可するとともに，ステータスレジスタの1STビットを'1'にセットします。

ビット4，5はステータスレジスタの1STビットをクリアするための制御ビットで，ビット4がタイマA，ビット5がタイマBの1STビットクリアに使用されます。このビットを'1'にすると，該当する1STビットがクリアされます。

ビット7はタイマAによる一斉キーオン機能を使うか否かを指定するもので，'1'を設定しておくと，タイマAのオーバフローが発生したときに自動的に全スロットがキーオンされます。

❷·❻7 LFO周波数設定レジスタ

274ページの図15にビット配置を示します。ビブラートなどをかけるためのLFOの周波数を決定します。設定値と周波数の関係を表1に示します。

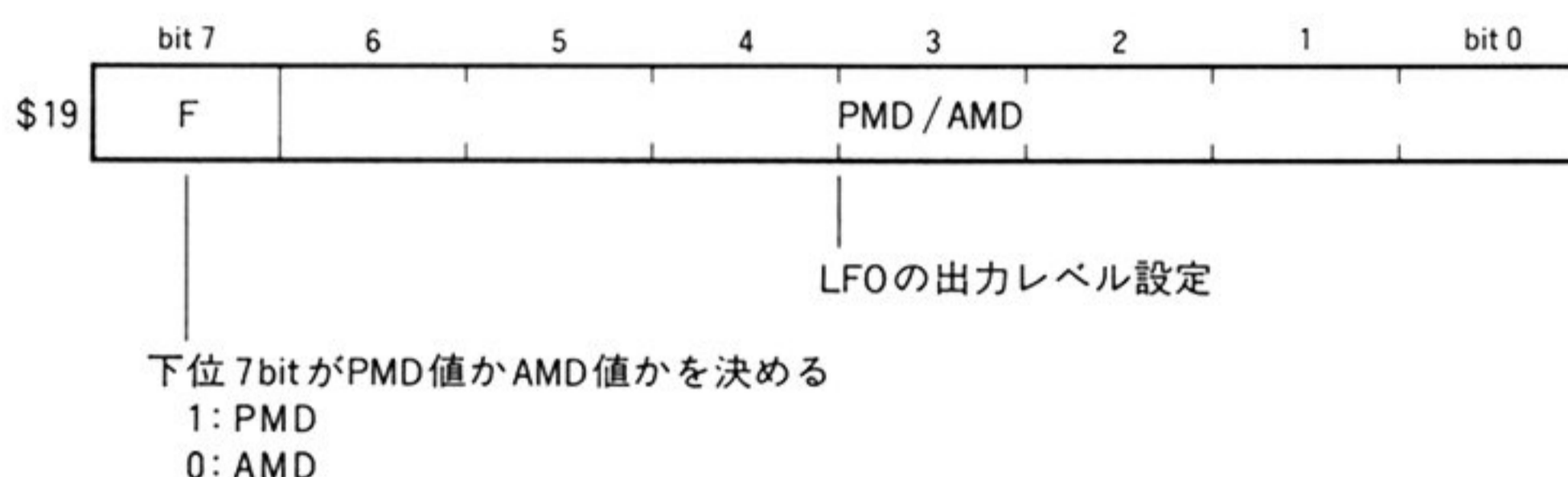
●図……15　LFO 周波数設定

●表……1　LFRQ の設定値と LFO の発振周波数の関係

DATA (HEX)	FREQ. (Hz)	DATA (HEX)	FREQ. (Hz)	DATA (HEX)	FREQ. (Hz)	DATA (HEX)	FREQ. (Hz)
FF	59.1278	BF	3.6955	7F	0.2310	3F	0.0144
FE	57.2205	BE	3.5763	7E	0.2235	3E	0.0140
FD	55.3131	BD	3.4571	7D	0.2161	3D	0.0135
FC	53.4058	BC	3.3379	7C	0.2086	3C	0.0130
FB	51.4984	BB	3.2187	7B	0.2012	3B	0.0126
FA	49.5911	BA	3.0994	7A	0.1937	3A	0.0121
F9	47.6837	B9	2.9802	79	0.1863	39	0.0116
F8	45.7764	B8	2.8610	78	0.1788	38	0.0112
F7	43.8690	B7	2.7418	77	0.1714	37	0.0107
F6	41.9617	B6	2.6226	76	0.1639	36	0.0102
F5	40.0543	B5	2.5034	75	0.1565	35	0.0098
F4	38.1470	B4	2.3842	74	0.1490	34	0.0093
F3	36.2396	B3	2.2650	73	0.1416	33	0.0088
F2	34.3323	B2	2.1458	72	0.1341	32	0.0084
F1	32.4249	B1	2.0265	71	0.1267	31	0.0079
F0	30.5176	B0	1.9073	70	0.1192	30	0.0075
EF	29.5639	AF	1.8477	6F	0.1155	2F	0.0072
EE	28.6102	AE	1.7881	6E	0.1118	2E	0.0070
ED	27.6566	AD	1.7285	6D	0.1080	2D	0.0068
EC	25.7029	AC	1.6689	6C	0.1043	2C	0.0065
EB	25.7492	AB	1.6093	6B	0.1006	2B	0.0063
EA	24.7955	AA	1.5497	6A	0.0969	2A	0.0061
E9	23.8419	A9	1.4901	69	0.0931	29	0.0058
E8	22.8882	A8	1.4305	68	0.0894	28	0.0056
E7	21.9345	A7	1.3709	67	0.0857	27	0.0054
E6	20.9808	A6	1.3113	66	0.0820	26	0.0051
E5	20.0272	A5	1.2517	65	0.0782	25	0.0049
E4	19.0735	A4	1.1921	64	0.0745	24	0.0047
E3	18.1198	A3	1.1325	63	0.0708	23	0.0044
E2	17.1661	A2	1.0729	62	0.0671	22	0.0042
E1	16.2125	A1	1.0133	61	0.0633	21	0.0040
E0	15.2588	A0	0.9537	60	0.0596	20	0.0037
DF	14.7820	9F	0.9239	5F	0.0577	1F	0.0036
DE	14.3051	9E	0.8941	5E	0.0559	1E	0.0035
DD	13.8283	9D	0.8643	5D	0.0540	1D	0.0034
DC	13.3514	9C	0.8345	5C	0.0522	1C	0.0033
DB	12.8746	9B	0.8047	5B	0.0503	1B	0.0031
DA	12.3978	9A	0.7749	5A	0.0484	1A	0.0030
D9	11.9209	99	0.7451	59	0.0466	19	0.0029
D8	11.4441	98	0.7153	58	0.0447	18	0.0028
D7	10.9673	97	0.6855	57	0.0428	17	0.0027
D6	10.4904	96	0.6557	56	0.0410	16	0.0026
D5	10.0136	95	0.6258	55	0.0391	15	0.0024
D4	9.5367	94	0.5960	54	0.0373	14	0.0023
D3	9.0599	93	0.5662	53	0.0354	13	0.0022
D2	8.5831	92	0.5364	52	0.0335	12	0.0021
D1	8.1062	91	0.5066	51	0.0317	11	0.0020
D0	7.6294	90	0.4768	50	0.0298	10	0.0019
CF	7.3910	8F	0.4619	4F	0.0289	0F	0.0018
CE	7.1526	8E	0.4470	4E	0.0279	0E	0.0017

CD	6.9141	8D	0.4321	4D	0.0270	0D	0.0017
CC	6.6757	8C	0.4172	4C	0.0261	0C	0.0016
CB	6.4373	8B	0.4023	4B	0.0251	0B	0.0016
CA	6.1989	8A	0.3874	4A	0.0242	0A	0.0015
C9	5.9605	89	0.3725	49	0.0233	09	0.0015
C8	5.7220	88	0.3576	48	0.0224	08	0.0014
C7	5.4836	87	0.3427	47	0.0214	07	0.0013
C6	5.2452	86	0.3278	46	0.0205	06	0.0013
C5	5.0068	85	0.3129	45	0.0196	05	0.0012
C4	4.7684	84	0.2980	44	0.0186	04	0.0012
C3	4.5300	83	0.2831	43	0.0177	03	0.0011
C2	4.2915	82	0.2682	42	0.0168	02	0.0010
C1	4.0531	81	0.2533	41	0.0158	01	0.0010
C0	3.8147	80	0.2384	40	0.0149	00	0.0009

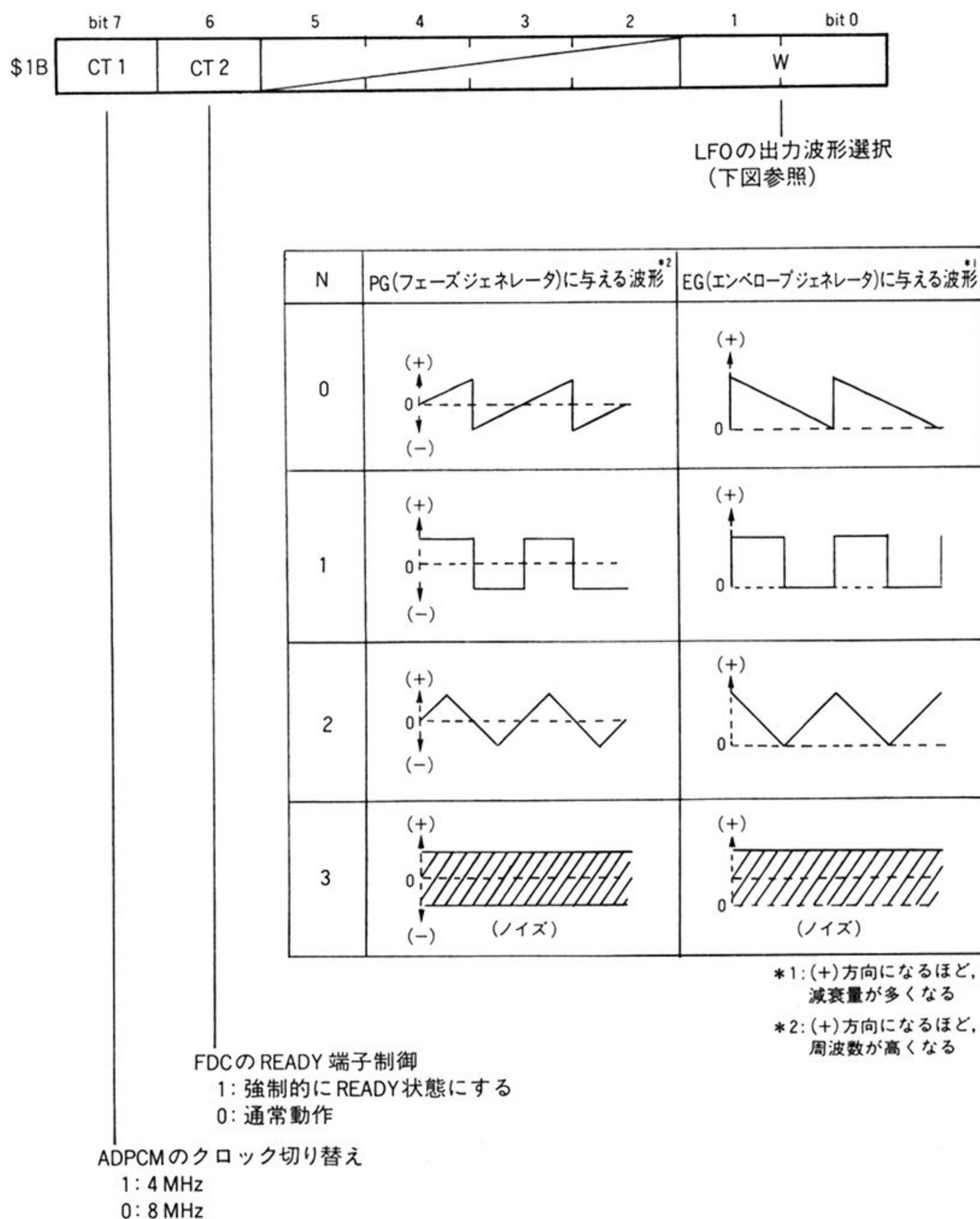
❷・❻ 8 PMD/AMD設定レジスタ

LFOが出力するPG用の出力，EG用の出力それぞれの出力レベルを設定するものです。このレジスタのビット配置を図16に示します。ビット7で書き込む値をPMDとするかAMDとするかを決定します。ビット7が'1'だと下位7ビットはPMD，'0'だとAMDとして扱われます。この値が大きくなるほど，出力レベルが大きくなり，深い変調がかかるようになります。

●図……16　PMD/AMD設定

❷・❻ 9 汎用出力/LFO出力波形制御レジスタ

ビット配置を276ページの図17に示します。OPMは各種の用途に使用できる汎用の出力端子を2つ持っていますが，X 68000では，これをFDC(フロッピーディスクコントローラ)のREADY端子を強制的にREADY状態にする信号，およびADPCMのクロック切り替え信号として使っています。

レジスタの下位2ビットはLFOの出力波形を選択するもので，図にあるような4種類の中

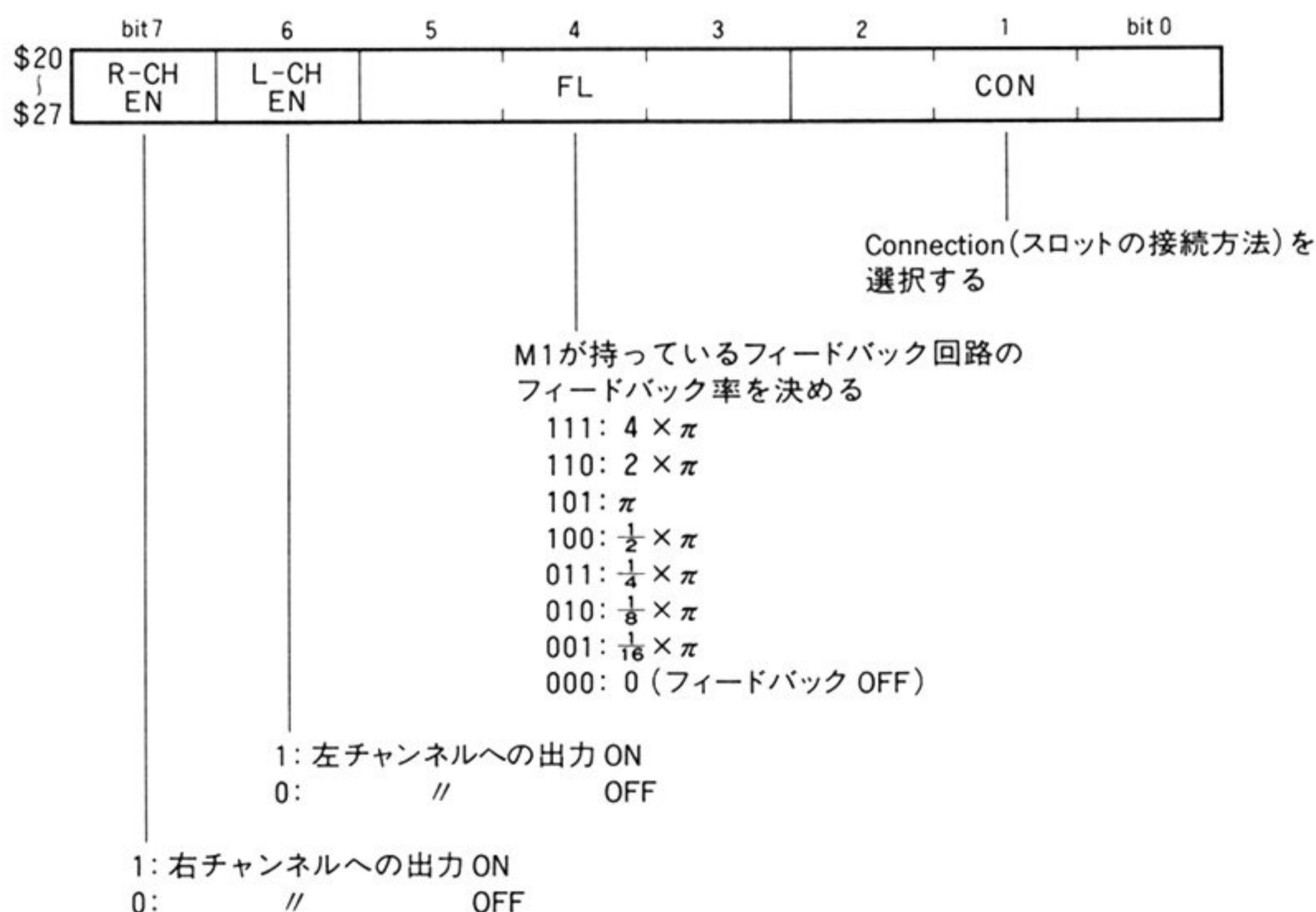
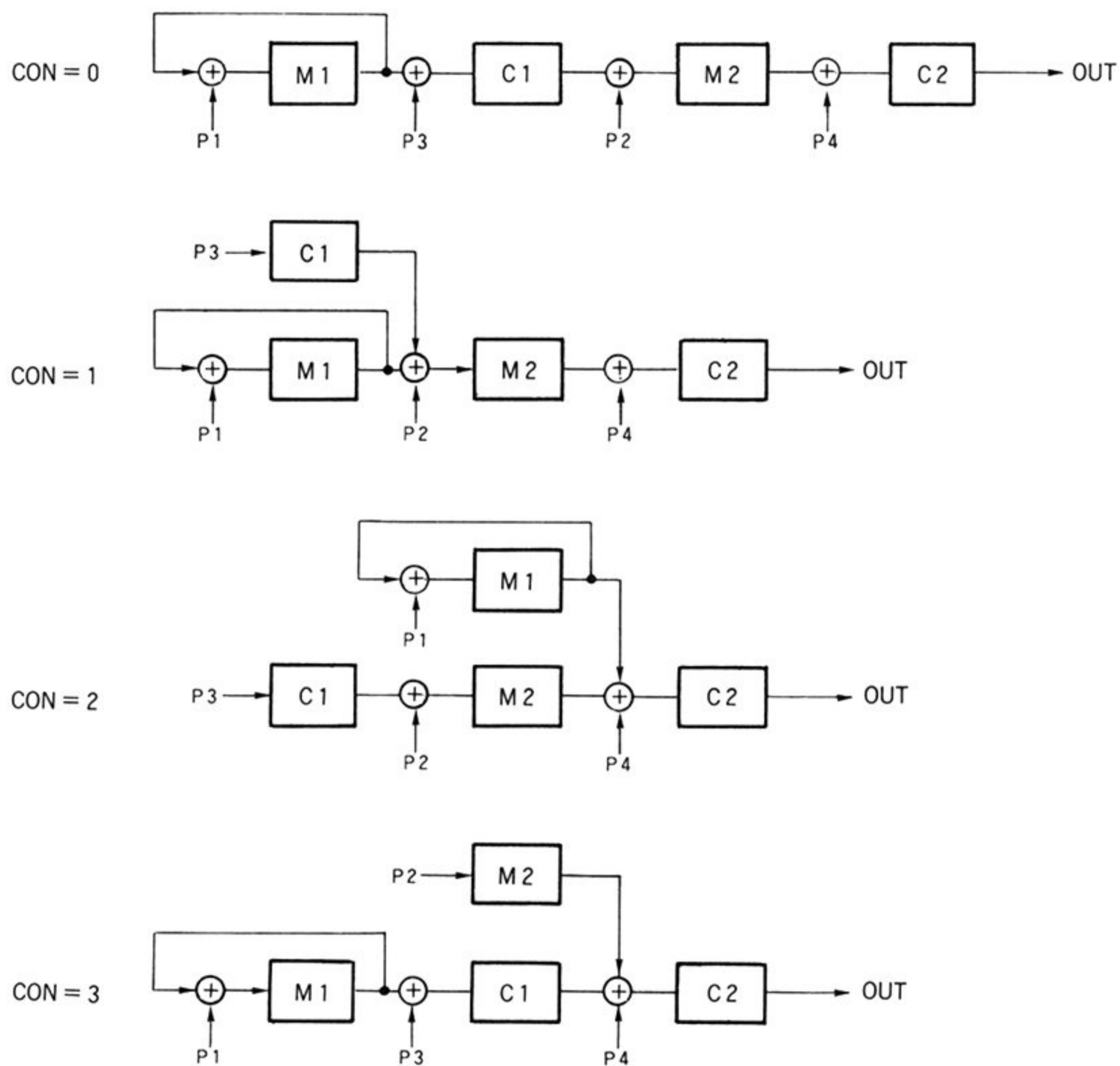
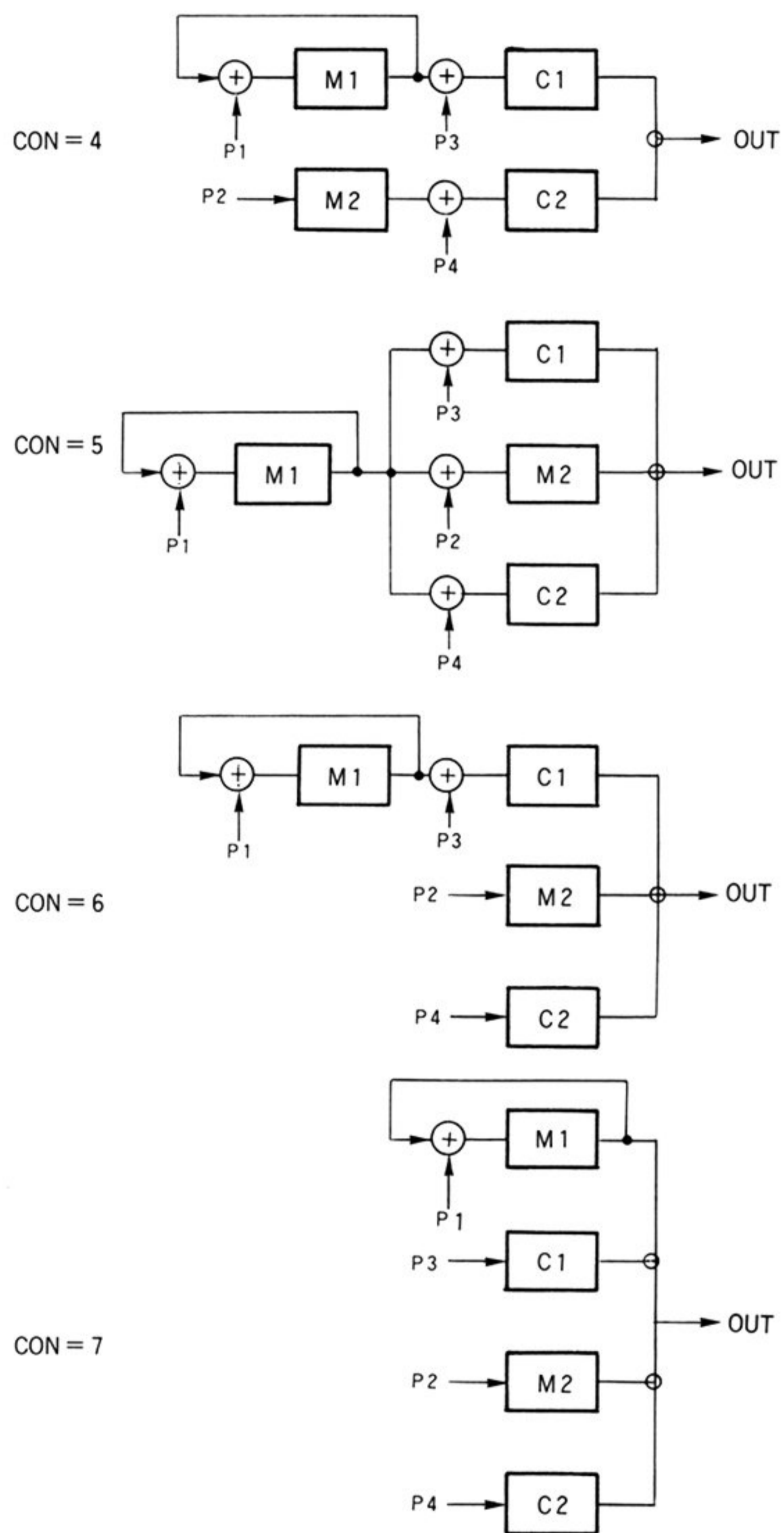
●図……17　汎用出力/LFO 出力波形制御

から選択することができます。Wを'11'（=3）に設定すると，LFOの出力はノイズ波形になり，EGやPGをランダムに振ることができるようになります。ノイズジェネレータが独立したノイズ音であるのに対し，LFOのノイズ出力は基本音の周波数や出力レベルをランダムに振るという点が異なります。

❷・❻10 チャンネル構成，出力制御レジスタ

スロットの接続方法や，M１スロットのフィードバック率，左右チャンネルへの出力選択を行うレジスタです。ビット配置は図18のようになっています。ビット６，７はそれぞれ左チャンネル，右チャンネルへの出力のON/OFFを制御するもので，'1'を書き込むとそれぞれのチャンネルへの出力がONに，'0'を書き込むとOFFになります。片方だけをONにするとそちら側から，両方ONにすると中央から音が出ているように感じます。

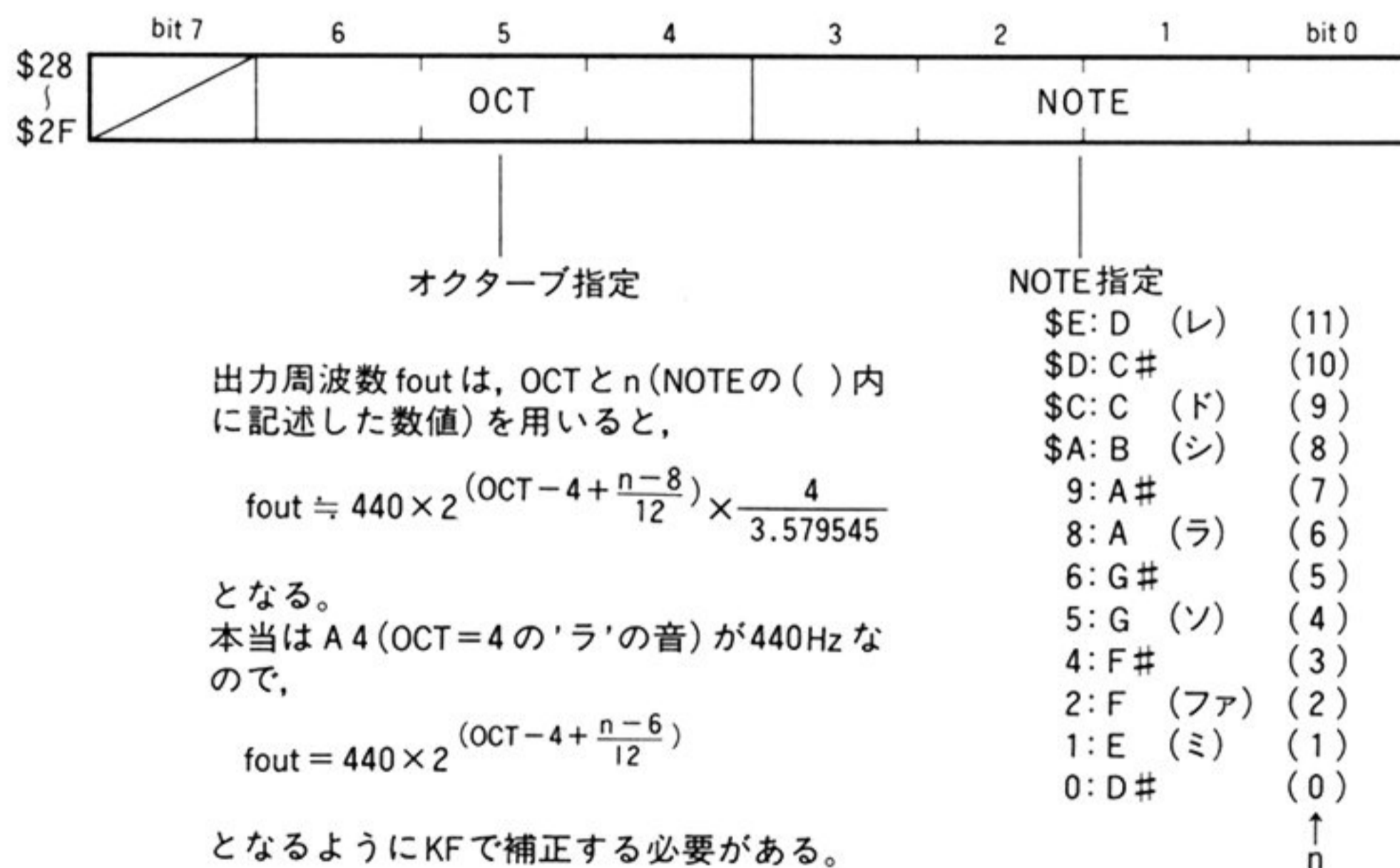
ビット３～５は，M１スロットが持っているフィードバック回路を通して帰還させる比率の選択を行うものです。選択するフィードバック率は，スロットの出力レベルにかけ算されるため，出力レベルが下がると，フィードバックされる量も少なくなります。

ビット０～２の３ビットはスロットの接続方法を決めるものです。スロットの接続形態は図19と図20に示すような８種類があり，この中からどれを使用するかを決めるわけです。

●図……18　チャンネル構成，出力制御（チャンネルごとに設定）

●図……19　コネクションの種類（その１）

●図……20　コネクションの種類（その2）

❷·❻11 KC（キーコード）レジスタ

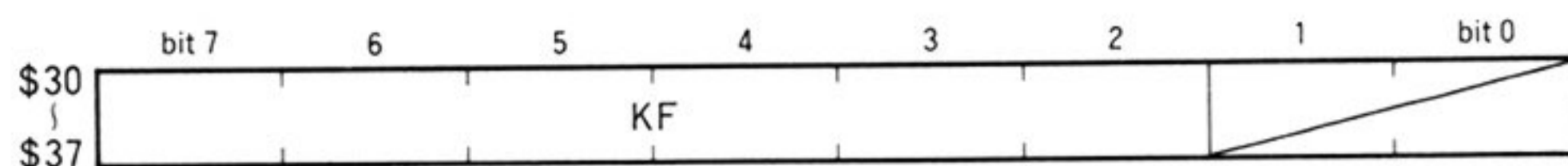
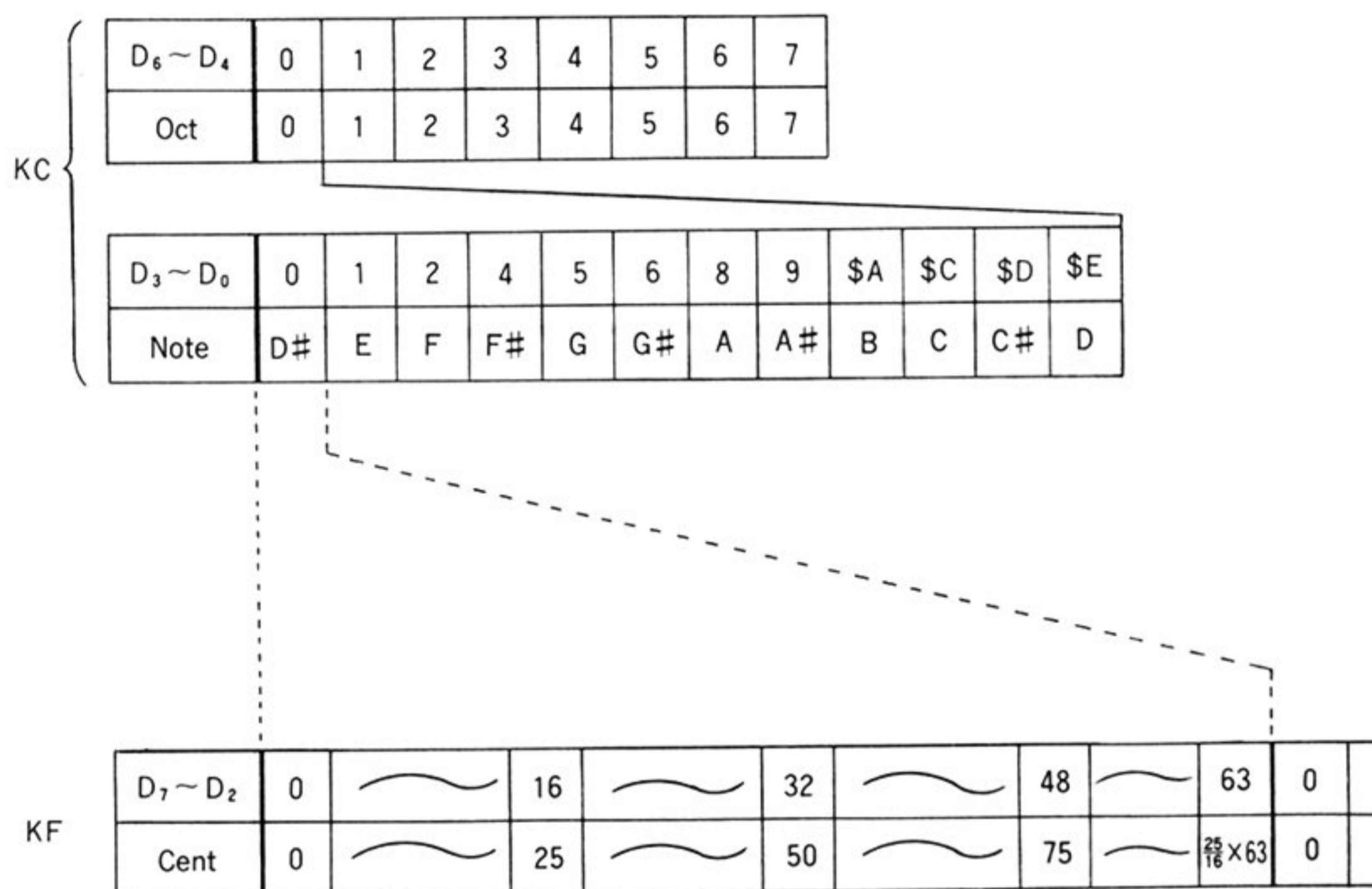
KCレジスタはチャンネルが動作する基本周波数を音階単位で選択します。ビット配置は図21のようになり，何オクターブ目の，何の音であるのかという設定を行います。OPMはクロック周波数として3.579545 MHzが与えられたときにもっとも誤差が少なくなるような音階テーブルを持っているのですが，X 68000ではクロックとして4 MHzを与えているため，約1音（192.27セント）高い音が出てしまいます。このため，OPMのマニュアルではNOTEが$Aのときに Aの音となるところが，X 68000ではBの音になります。

音のずれが200セントぴったりではないため，気になる方は誤差分をKFレジスタで調整するようにしてください。

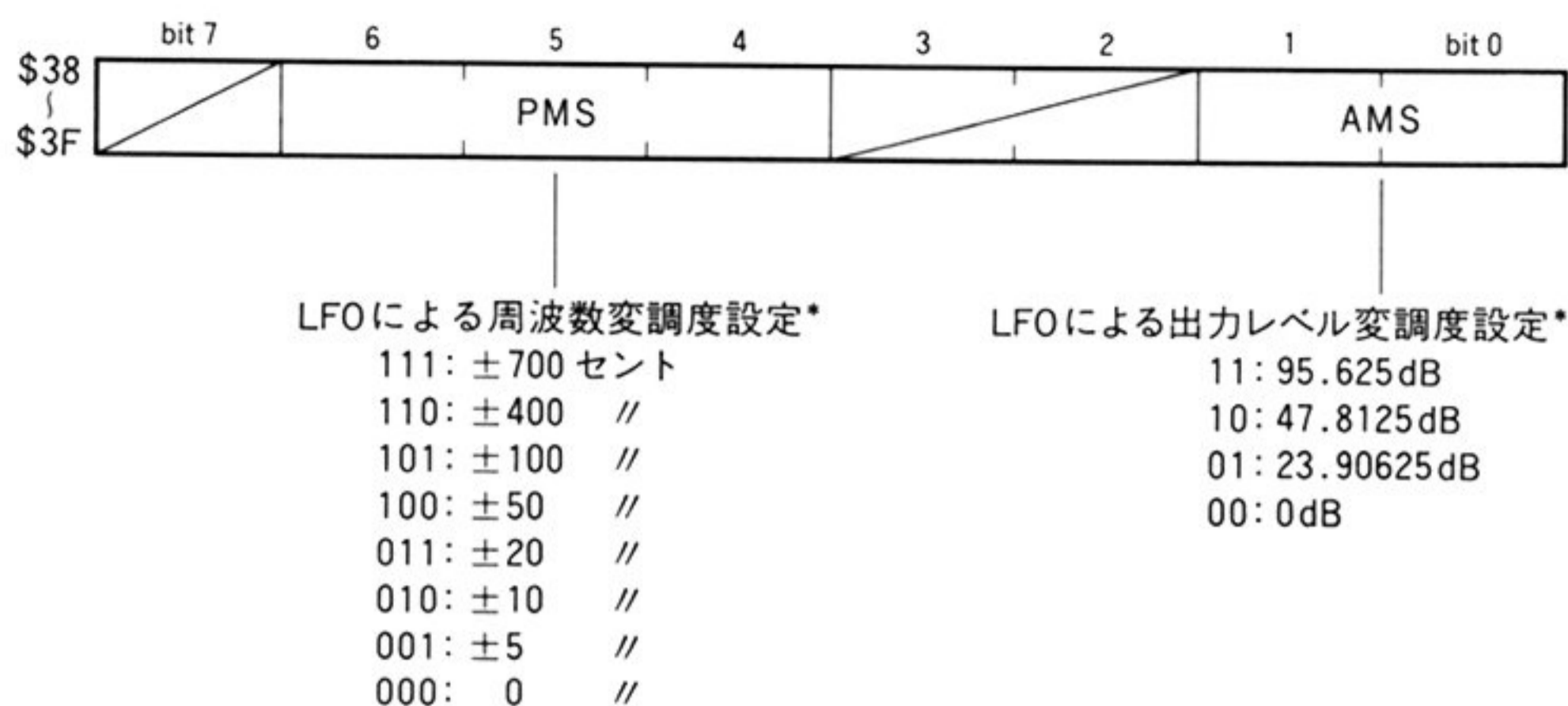
●図……21 KC（キーコード）（チャンネルごとに設定）

❷·❻12 KF（キーフラクション）レジスタ

レジスタのビット配置を図22に示します。

上位の6ビットだけが使用され，この値によってKCで設定した周波数に約1.6(＝100/64)セント単位の微調整を加えます。KCとKFの関係を図23に示しますので参考にしてください。

●図……22　KF(キーフラクション)（チャンネルごとに設定）

●図……23　KC，KF による音程指定

❷・❻13　PMS/AMS設定レジスタ

PMD/AMD が LFO の出力レベルを設定するのに対し，PMS/AMS はチャンネルごとに LFO の効きぐあいを設定するものです。レジスタのビット配置は 282 ページの図 24 のようになっています。ビット 4～6 は PG への効き方を，ビット 0，1 は EG への効きぐあいを設定します。図の中に書いた数値は，LFO の出力が最大レベルになったときの値を示しています。

●図……24　PMS/AMS（チャンネルごとに設定）

* 値はいずれもLFOの出力レベルが最大のときの値

❷・❻14　DT1/MUL設定レジスタ

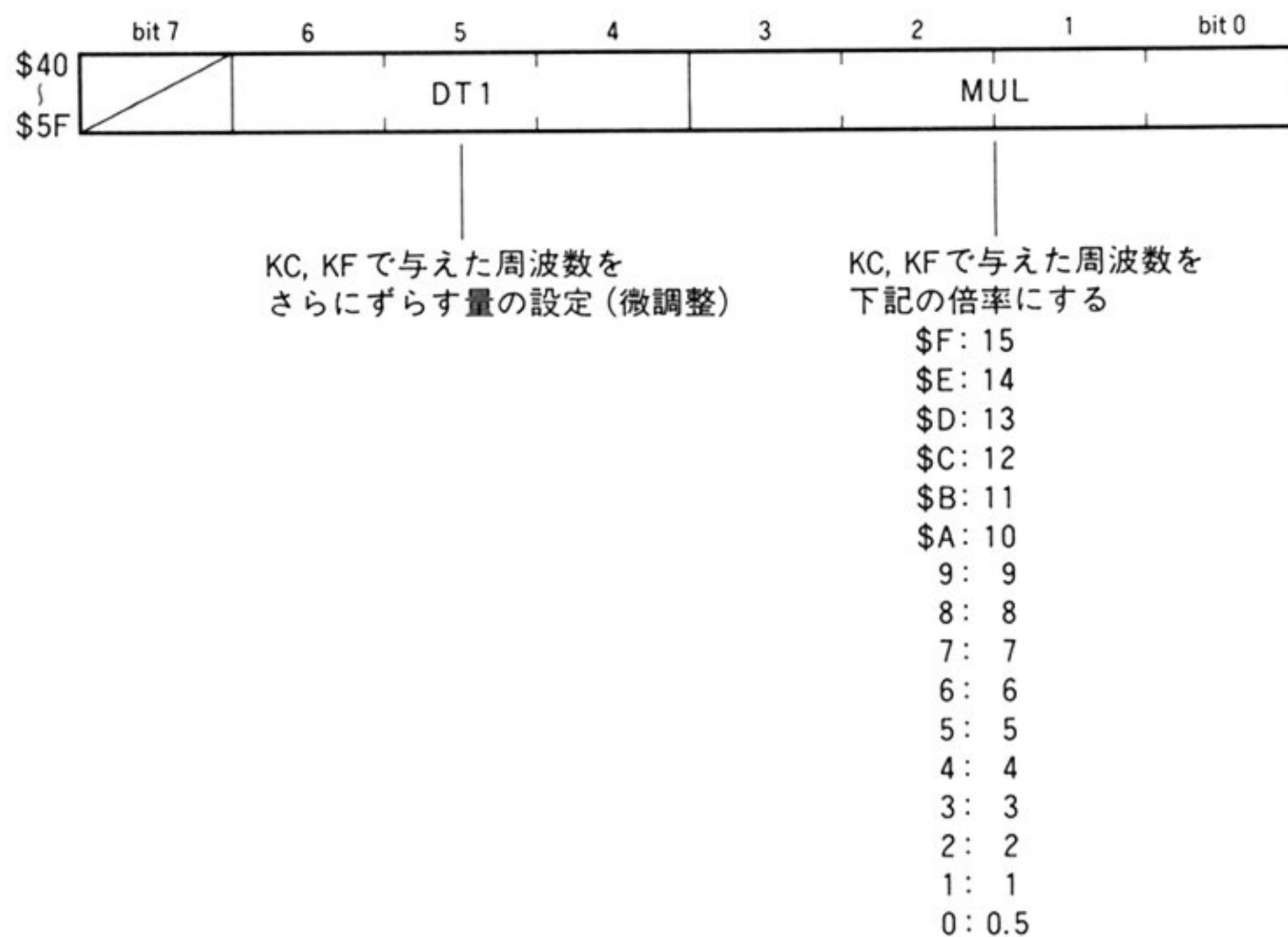
ビット配置を図25に示します。MUL，DT1およびDT2（$C0～$DFにあります）は，KCやKFで与えた周波数（仮りに基本周波数と呼ぶことにします）をもとにスロットごとに異なる周波数をつくり出すもので，MULが1/2～15倍までの倍率設定，DT1が数セント程度の微調整，DT2が数百セントレベルの大きなずれをつくるのに使用されます。

ビット0～3の4ビットは倍率設定で，0のときには基本周波数の1/2，1以上のときには基本周波数の設定値倍の周波数がPGから出力されます（DT1=DT2=0のとき）。

DT1による周波数変動はKCの値によって変わります。この関係を表2に示しますので参考にしてください。

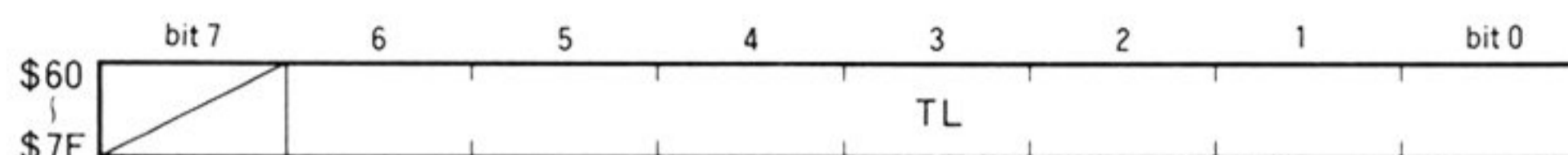
❷・❻15　TL（トータルレベル）設定

TLはスロットの出力レベルを制御するもので，EGで計算した出力に，このレジスタで決めた減衰量をかけて実際のEGの出力としています。レジスタのビット配置を284ページの図26に示します。TLレジスタは下位7ビットだけが有効で，減衰量は0.75×TL(dB)となります。TLが0のときがもっとも出力信号レベルが大きくなり，$7Fのときにもっとも小さくなります。

●図……25 DT1/MUL（スロットごとに設定）

●表……2 DT1の設定値と周波数のずれ方

OCT	NOTE	周波数のずれ(セント)				周波数のずれ(Hz)			
		DT1=0	DT1=1	DT1=2	DT1=3	DT1=0	DT1=1	DT1=2	DT1=3
0	0	0.000	0.000	5.025	10.036	0.000	0.000	0.053	0.107
0	1	0.000	0.000	4.228	8.445	0.000	0.000	0.053	0.107
0	2	0.000	0.000	3.559	7.110	0.000	0.000	0.053	0.107
0	3	0.000	0.000	2.993	5.980	0.000	0.000	0.053	0.107
1	0	0.000	2.515	5.025	5.025	0.000	0.053	0.107	0.107
1	1	0.000	2.115	4.228	6.338	0.000	0.053	0.107	0.160
1	2	0.000	1.778	3.555	5.330	0.000	0.053	0.107	0.160
1	3	0.000	1.496	2.990	4.483	0.000	0.053	0.107	0.160
2	0	0.000	1.258	2.515	5.025	0.000	0.053	0.107	0.213
2	1	0.000	1.057	3.170	4.225	0.000	0.053	0.160	0.213
2	2	0.000	0.889	2.667	3.555	0.000	0.053	0.160	0.213
2	3	0.000	0.748	2.242	3.735	0.000	0.053	0.160	0.267
3	0	0.000	1.258	2.515	3.143	0.000	0.107	0.213	0.267
3	1	0.000	1.057	2.114	3.170	0.000	0.107	0.213	0.320
3	2	0.000	0.889	1.778	2.667	0.000	0.107	0.213	0.320
3	3	0.000	0.748	1.869	2.615	0.000	0.107	0.267	0.373
4	0	0.000	0.629	1.572	2.515	0.000	0.107	0.267	0.427
4	1	0.000	0.793	1.586	2.114	0.000	0.160	0.320	0.427
4	2	0.000	0.667	1.334	2.001	0.000	0.160	0.320	0.480
4	3	0.000	0.561	1.308	1.869	0.000	0.160	0.373	0.533
5	0	0.000	0.629	1.258	1.729	0.000	0.213	0.427	0.587
5	1	0.000	0.529	1.057	1.586	0.000	0.213	0.427	0.640
5	2	0.000	0.445	1.001	1.445	0.000	0.213	0.480	0.693
5	3	0.000	0.467	0.935	1.308	0.000	0.267	0.533	0.747

6	0	0.000	0.393	0.865	1.258	0.000	0.267	0.587	0.853
6	1	0.000	0.397	0.793	1.123	0.000	0.320	0.640	0.907
6	2	0.000	0.334	0.723	1.056	0.000	0.320	0.693	1.013
6	3	0.000	0.327	0.654	0.935	0.000	0.373	0.747	1.067
7	0	0.000	0.315	0.629	0.865	0.000	0.427	0.853	1.173
7	1	0.000	0.315	0.629	0.865	0.000	0.427	0.853	1.173
7	2	0.000	0.315	0.629	0.865	0.000	0.427	0.853	1.173
7	3	0.000	0.315	0.629	0.865	0.000	0.427	0.853	1.173

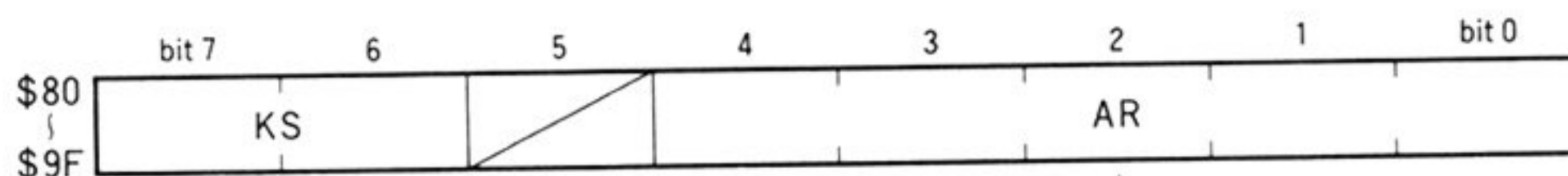
●図……26　TL（トータルレベル）（スロットごとに設定）

❷・❻ 16　KS/AR設定レジスタ

ビット配置を図27に示します。楽器の場合，音が高くなるほど音の立ち上がりや余韻の時間が短くなる傾向があります。この効果を得るためにあるのがKSで，ARや後述するD1R, D2R, RRなどにはすべてKSによる補正がかけられます。この補正量は，KC（キーコード）に依存し，(KC%4)%(2^(3−KS))（%は整数除算，'^'はべき乗を表す）となります。

ARはアタックレートで，キーオンした直後の音の立ち上がりを決めるものです。値が大きいほど立ち上がりが鋭くなります。

具体的な数値は，音量が最小（0%：−96 dB）から最大（100%：0 dB）になるまでの時間と，10%から90%になるまでの時間の2通りについて，それぞれ289ページ以降の表3と表4に示しておきましたので参考にしてください。

なお，この表のRATEの値は，RRの場合はRR×4+2+KS，それ以外のものの場合はレジスタへの設定値をXRとすると，XR×2+KSで算出される値を使用し，計算値が63以上になった場合にはRRが63のときの値を適用します。表の左端はRRの計算値，その右の2つの数値は，RRを上位4ビットと下位に分けたときの値を示しています。

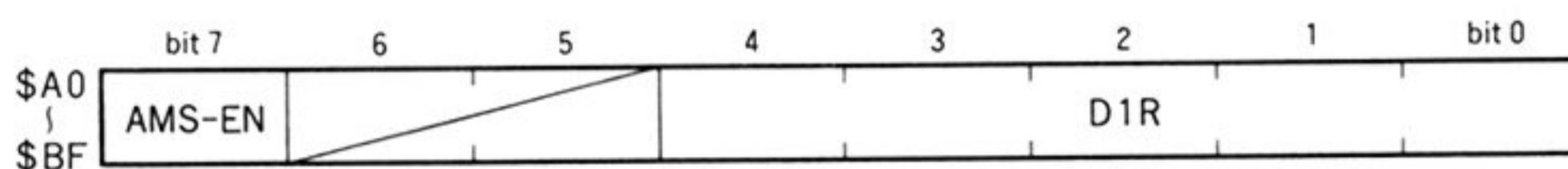
●図……27　KS/AR

$$\frac{\frac{KC}{4}}{2^{(3-KS)}}$$（ただし，除算時の小数点以下はすべて切り捨て）

を加える

❷·❻ 17　AMS-EN/D1R設定レジスタ

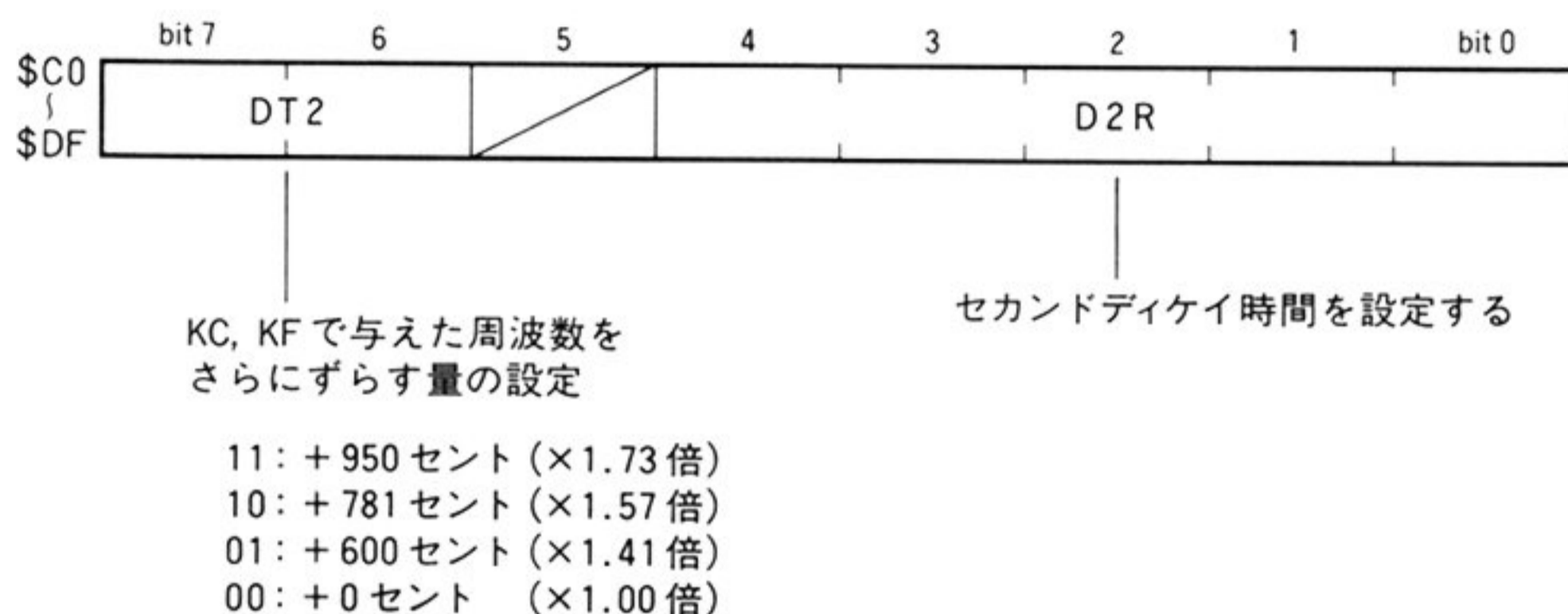
ビット配置を図 28 に示します。AMS-EN は LFO による EG の変調をかけるか否かを決めるものです。'1'を書き込むと変調がかかるようになり，'0'で通常どおりの動作となります。PG への変調は，チャンネル単位で決められてしまいますが，EG への変調は，このビットを使ってスロットごとにかけるか否かを決定できます。

D 1 R はファーストディケイからセカンドディケイに移るまでの時間を決めるもので，KS によってスケーリングされます。AR と同様，表にまとめておきましたので参考にしてください。

●図……28　AMS-EN/D1R（スロットごとに設定）

❷·❻18 DT2/D2R設定レジスタ

このレジスタのビット配置は図29のようになっています。DT 2はDT 1/MULのところでも触れたように，KC, KFで決まる周波数に数百セントという，大きめの変化を与えるものです。

D 2 Rはセカンドディケイレートで，キーオンしたままにしたとき，ファーストディケイからセカンドディケイに移った時点から音が完全に消えるまでの時間を設定します。この時間もD 1 R同様，KSによってスケーリングされます。

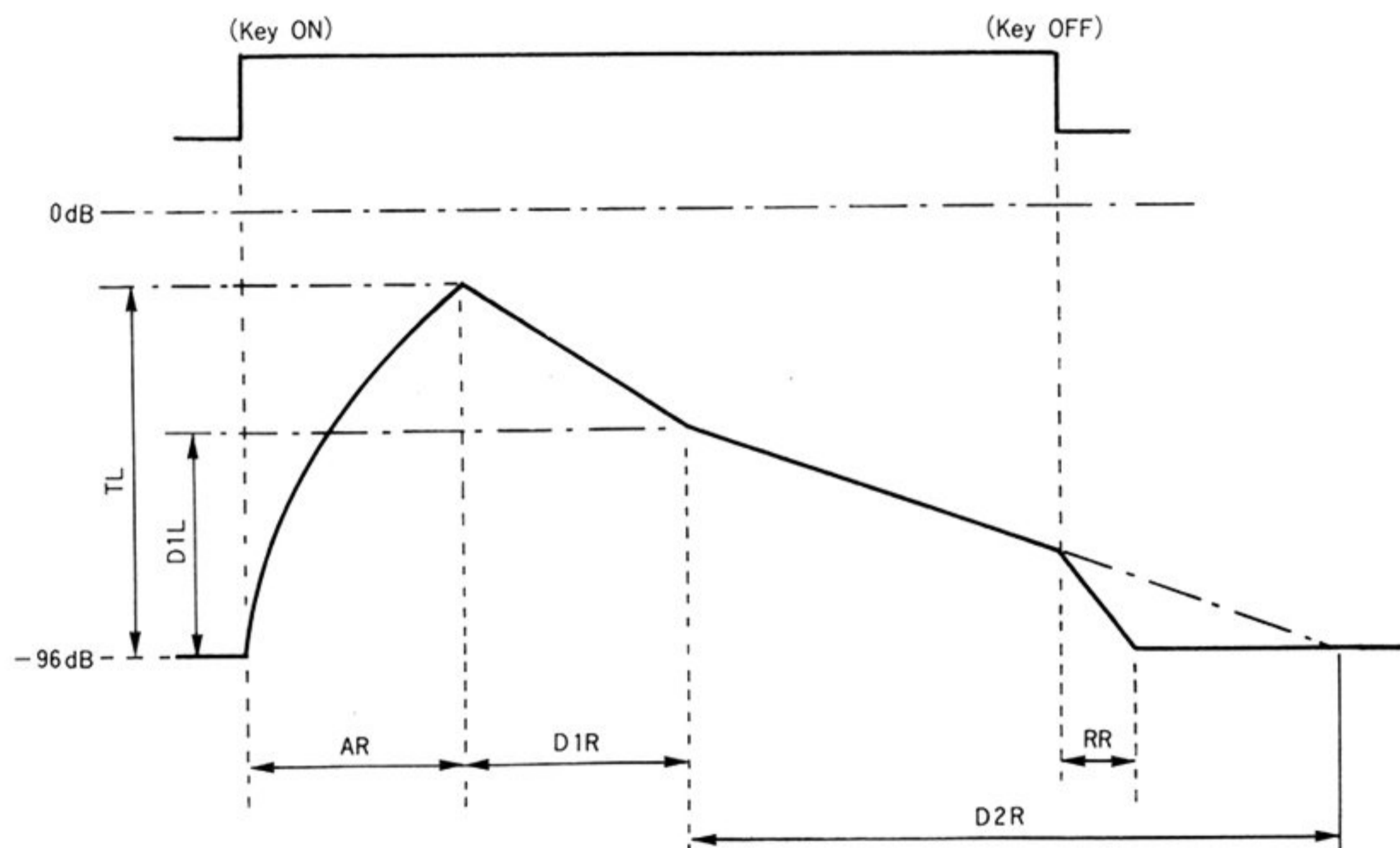
●図……29　DT2/D2R（スロットごとに設定）

❷·❻19 D1L/RR

ビット配置を図30に示します。上位4ビットがD1L，下位4ビットがRRになっています。D1Lはファーストディケイからセカンドディケイに移るときのレベルを決定するもので，値が0～$Eのときには，このレベルは−3×D1L(dB)，$Fのときには−3×D1L−48(dB)で表されます。

RRはリリースレートで，キーオフしてから音が消えるまでの時間の設定を行います。RRもAR，D1R，D2Rなどと同様，KSによってスケーリングされます。

EGの出力波形と，レジスタへの設定値の関係を図31に示しておきましたので参考にしてください。なお，この図ではKSによるスケーリングは考慮していません。

●図……30　D1L/RR（スロットごとに設定）

●図……31　エンベロープジェネレータの出力波形と設定値の関係

❷·7　設定値とOPMの動作の関係

OPMの設定値が実際の音声出力波形にどのように影響するかを数値で示した資料は，残念ながら公開されていません(メーカでも公表する予定はないとのことでした)。たしかに，音作りの面から見ると，パラメータの調整はあくまでも耳で聴きながら行うものであって，出力波形を見ながら行うようなものではありませんが，X 68000のように，ADPCMも搭載しているマシンでは，FM音源用のパラメータを使ってADPCMから出力するなどの使い方も考えら

れますので，音作りの基本であるオペレータの直列接続と，EGの各レートの計算方法を個人的に調べてみました。

以下に示す式は，あくまでも筆者が個人的に（シンクロで波形を見ながら）調べた近似式であり，OPMがこのとおりにつくられているということではありませんので，注意してください。

❷·❼1 オペレータを直列接続したときの出力波形

オペレータを2つ直列に接続したときのSIN波形テーブルの出力Aは，

$$A=SIN(\omega t+\alpha SIN(\psi t))$$

のように表されます。ψt, ωtは，要するに周波数ですから，これはかんたんに求められますが，αの求め方が公開されていないので，このままでは出力波形がわかりません。実験の結果，各スロットの出力がTLだけで決まるとき（ARが0，D1R，D2Rなどが最大値のとき）には，

$$\alpha=10\hat{}(-0.75/20\times TL+e/2)$$

ここで，eは自然対数の底（=2.71828…），'^'はべき乗を示す

とし，$\alpha SIN(\psi t)$の計算結果の数値をラジアン単位の角度データとみなしてωtと足すと，かなりよい感じになることがわかりました。

M 1のフィードバック量はよくわかりませんでしたが，TLが0 dBのときに1（ラジアン）とみなしてフィードバックするとよいようです。たとえば，FL=3なら0.7853(=3.14159/4)をかけ算した値をフィードバックすると，近い波形が得られます。

❷·❼2 EGのRATEと時間の関係

EGの表の値はほぼ指数関数となっています。0～100%のアタック時間tは，RATEの上位4ビットと下位2ビット（表の2番目，3番目の数値）を使って計算されます。上位4ビットを$RATE_H$，下位2ビットを$RATE_L$とすると，

$$t=(10\hat{}4.202682)/(2\hat{}RATE_H)\times(1/(1+0.25\times RATE_L))\times 3.58/4.00$$

で表されます。その他の時間は，この値にたんに係数をかけるだけで算出することができます。OPM の時間の分解能の限界のために，t が小さくなってくると，この式で計算した値からずれてしまいますので注意してください。

●表……3　EG の各レート設定値と時間（0～100 %）

アタック		ファーストディケイ/セカンドディケイ/リリース	
AR×2+KS	時間(ms)	XR×2+KS RR×4+2+KS	時間(ms)
0 : 0 0	無限大	0 : 0 0	無限大
1 : 0 1	無限大	1 : 0 1	無限大
2 : 0 2	無限大	2 : 0 2	無限大
3 : 0 3	無限大	3 : 0 3	無限大
4 : 1 0	7136.33	4 : 1 0	98637.69
5 : 1 1	5709.06	5 : 1 1	78910.15
6 : 1 2	4757.55	6 : 1 2	65758.46
7 : 1 3	4077.90	7 : 1 3	56364.40
8 : 2 0	3568.16	8 : 2 0	49318.84
9 : 2 1	2854.53	9 : 2 1	39455.07
10 : 2 2	2378.78	10 : 2 2	32879.23
11 : 2 3	2038.95	11 : 2 3	28182.20
12 : 3 0	1784.08	12 : 3 0	24659.42
13 : 3 1	1427.27	13 : 3 1	19727.54
14 : 3 2	1189.38	14 : 3 2	16439.61
15 : 3 3	1019.48	15 : 3 3	14091.09
16 : 4 0	892.04	16 : 4 0	12329.71
17 : 4 1	713.63	17 : 4 1	9863.77
18 : 4 2	594.69	18 : 4 2	8219.81
19 : 4 3	509.74	19 : 4 3	7045.55
20 : 5 0	446.02	20 : 5 0	6164.86
21 : 5 1	356.82	21 : 5 1	4931.89
22 : 5 2	297.35	22 : 5 2	4109.90
23 : 5 3	254.87	23 : 5 3	3522.77
24 : 6 0	223.01	24 : 6 0	3082.42
25 : 6 1	178.41	25 : 6 1	2465.94
26 : 6 2	148.68	26 : 6 2	2054.95
27 : 6 3	127.43	27 : 6 3	1761.38
28 : 7 0	111.51	28 : 7 0	1541.22
29 : 7 1	89.20	29 : 7 1	1232.97
30 : 7 2	74.34	30 : 7 2	1027.48
31 : 7 3	63.72	31 : 7 3	880.59
32 : 8 0	55.75	32 : 8 0	770.60
33 : 8 1	44.60	33 : 8 1	616.48
34 : 8 2	37.16	34 : 8 2	513.74
35 : 8 3	31.86	35 : 8 3	440.35
36 : 9 0	27.88	36 : 9 0	385.31
37 : 9 1	22.30	37 : 9 1	308.25
38 : 9 2	18.58	38 : 9 2	256.83
39 : 9 3	15.93	39 : 9 3	220.17
40 : 10 0	13.94	40 : 10 0	192.65
41 : 10 1	11.15	41 : 10 1	154.12
42 : 10 2	9.29	42 : 10 2	128.43
43 : 10 3	7.97	43 : 10 3	110.09
44 : 11 0	6.97	44 : 11 0	96.33
45 : 11 1	5.58	45 : 11 1	77.06
46 : 11 2	4.65	46 : 11 2	64.22
47 : 11 3	3.98	47 : 11 3	55.04
48 : 12 0	3.48	48 : 12 0	48.16
49 : 12 1	2.78	49 : 12 1	38.53
50 : 12 2	2.33	50 : 12 2	32.11
51 : 12 3	1.99	51 : 12 3	27.52
52 : 13 0	1.91	52 : 13 0	24.08
53 : 13 1	1.53	53 : 13 1	19.27
54 : 13 2	1.27	54 : 13 2	16.06
55 : 13 3	1.09	55 : 13 3	13.77

56 : 14 0	1.00	56 : 14 0	12.04
57 : 14 1	0.81	57 : 14 1	9.63
58 : 14 2	0.67	58 : 14 2	8.03
59 : 14 3	0.57	59 : 14 3	6.88
60 : 15 0	0.47	60 : 15 0	6.02
61 : 15 1	0.47	61 : 15 1	6.02
62 : 15 2	0.47	62 : 15 2	6.02
63 : 15 3	0.00	63 : 15 3	6.02

●表……4　EGの各レート設定値と時間（10〜90％）

アタック		ファーストディケイ/セカンドディケイ/リリース	
AR×2＋KS	時間（ms）	XR×2＋KS RR×4＋2＋KS	時間（ms）
0 : 0 0	無限大	0 : 0 0	無限大
1 : 0 1	無限大	1 : 0 1	無限大
2 : 0 2	無限大	2 : 0 2	無限大
3 : 0 3	無限大	3 : 0 3	無限大
4 : 1 0	4008.07	4 : 1 0	19942.60
5 : 1 1	3206.45	5 : 1 1	15954.08
6 : 1 2	2672.05	6 : 1 2	13295.07
7 : 1 3	2290.32	7 : 1 3	11395.78
8 : 2 0	2004.04	8 : 2 0	9971.30
9 : 2 1	1603.23	9 : 2 1	7977.05
10 : 2 2	1336.02	10 : 2 2	6647.53
11 : 2 3	1145.16	11 : 2 3	5697.88
12 : 3 0	1002.02	12 : 3 0	4985.65
13 : 3 1	801.62	13 : 3 1	3988.52
14 : 3 2	668.01	14 : 3 2	3323.77
15 : 3 3	572.59	15 : 3 3	2848.95
16 : 4 0	501.01	16 : 4 0	2492.83
17 : 4 1	400.81	17 : 4 1	1994.26
18 : 4 2	334.01	18 : 4 2	1661.88
19 : 4 3	286.29	19 : 4 3	1424.47
20 : 5 0	250.50	20 : 5 0	1246.41
21 : 5 1	200.40	21 : 5 1	997.13
22 : 5 2	167.01	22 : 5 2	830.94
23 : 5 3	143.15	23 : 5 3	712.23
24 : 6 0	125.26	24 : 6 0	623.21
25 : 6 1	100.20	25 : 6 1	498.57
26 : 6 2	83.50	26 : 6 2	415.47
27 : 6 3	71.57	27 : 6 3	356.12
28 : 7 0	62.62	28 : 7 0	311.60
29 : 7 1	50.10	29 : 7 1	249.28
30 : 7 2	41.75	30 : 7 2	207.74
31 : 7 3	35.78	31 : 7 3	178.04
32 : 8 0	31.32	32 : 8 0	155.80
33 : 8 1	25.05	33 : 8 1	124.64
34 : 8 2	20.87	34 : 8 2	103.86
35 : 8 3	17.89	35 : 8 3	89.03
36 : 9 0	15.65	36 : 9 0	77.90
37 : 9 1	12.52	37 : 9 1	62.32
38 : 9 2	10.44	38 : 9 2	52.83
39 : 9 3	8.95	39 : 9 3	44.52
40 : 10 0	7.83	40 : 10 0	38.95
41 : 10 1	6.27	41 : 10 1	31.16
42 : 10 2	5.22	42 : 10 2	25.96
43 : 10 3	4.48	43 : 10 3	22.26
44 : 11 0	3.91	44 : 11 0	19.48
45 : 11 1	3.13	45 : 11 1	15.58
46 : 11 2	2.61	46 : 11 2	12.98
47 : 11 3	2.24	47 : 11 3	11.12
48 : 12 0	1.96	48 : 12 0	9.74
49 : 12 1	1.57	49 : 12 1	7.79
50 : 12 2	1.31	50 : 12 2	6.49
51 : 12 3	1.12	51 : 12 3	5.57
52 : 13 0	0.98	52 : 13 0	4.87
53 : 13 1	0.78	53 : 13 1	3.89

54：13 2	0.65	54：13 2	3.26
55：13 3	0.55	55：13 3	2.78
56：14 0	0.52	56：14 0	2.43
57：14 1	0.42	57：14 1	1.95
58：14 2	0.36	58：14 2	1.62
59：14 3	0.30	59：14 3	1.39
60：15 0	0.24	60：15 0	1.22
61：15 1	0.24	61：15 1	1.22
62：15 2	0.24	62：15 2	1.22
63：15 3	0.00	63：15 3	1.22

●3 ADPCM

❸·1 ADPCMの概要

OPM（FM音源）がSINカーブやノイズなどをもとに演算で波形を作成するのに対し，ADPCMのほうは自然音の取り込みや再生を行うものです。

入力波形を定期的にサンプリングして，その時点での電圧を，そのままデジタルデータに変換するのが，CDなどでも採用されているPCM方式と呼ばれるものです。PCM方式ではサンプリング周波数のほかにはなんら制約はありませんから，波形の再現性はよいのですが，メモリを大量に食うという問題があります。たとえば，CDと同様にすると，1回分のデータが16ビット，サンプリング周波数44.1KHzですから，1秒で88.2Kバイトも使ってしまいます。

なんとか，この量を減らそうと考えられた方法に，前回の電圧との差分をデータ化する⊿PCM法，前回の変化から次のデータを予測し，そのデータとの差分をデータ化するDPCM（Differential-PCM）法などがあります。ADPCM（Adaptive Differential PCM）はDPCMをさらに改良して，大きな電圧の変化にも対応できるようにし，音質を改善したものです。前回の変化が大きいときには次の変化の幅も大きく，小さいときには変化幅も小さいものと考え，前回の変化の大きさから求められる定数を，予測値との変化にかけた値をデータ化しようというものです。

X68000では，入力波形からADPCMデータへの変換やADPCMデータから出力波形の再現を行うLSIとして，沖電気のMSM6258VというLSIを使用しています。このLSIは，入出力ともモノラルであるため，X68000ではパンポット制御（右，左，中央のいずれから出力するかを選択する）回路を付加しています。

次にMSM6258Vの特徴をかんたんにまとめておきます。

❸·❶1 ADPCMデータ

MSM 6258 V は，ADPCM データとして 3 ビットまたは 4 ビットのデータを作成し，サンプリング 2 回分のデータをまとめて 1 バイトデータにして CPU とやりとりするようになっています。3 ビット ADPCM とするか，4 ビット ADPCM とするかは，LSI のピンで切り替えられるのですが，X 68000 では 4 ビット ADPCM モードに固定して使用しています。

❸·❶2 サンプリング周波数

サンプリング周波数は，LSI のクロック周波数の 1/512，1/768，1/1024 のいずれかから選択可能です。X 68000 ではクロックとして 4 MHz と 8 MHz を切り替えられるようにしているため，3.9 KHz，5.2 KHz，7.8 KHz，10.4 KHz，15.6 KHz の 5 種類を選択できます(4 MHz で 1/512 のときと，8 MHz で 1/1024 のときはどちらも 7.8 KHz になるため，1 種類減ります)。1 秒あたり使用するメモリ量は，サンプリング周波数の半分(15.6 KHz なら 7.8 K バイト）になります。

❸·❶3 A/D, D/Aコンバータ

MSM 6258 V に内蔵されている A/D コンバータ（入力電圧からデジタルデータへの変換器）は 8 ビット，D/A コンバータ（デジタルデータから出力電圧への変換器）は 10 ビットの精度を持っています。MSM 6258 V は，入力データを 8 ビットの PCM データに直した後，ADPCM 変換を行い，また ADPCM データを 10 ビットの PCM データに変換した後，音声信号として出力しているわけです。

❸·2 ADPCM関係のレジスタ

ADPCM の制御に関係するレジスタの一覧を図 32 に示します。MSM 6258 V が持っているレジスタには最低限のステータスやコマンドしかないため，X 68000 ではサンプリングレートや ADPCM の出力切り替えを PPI (8255) のポート C で，ADPCM の基本クロックの選択を OPM の汎用出力端子 CT 2 で行えるようにしています。

●図……32　ADPCMの動作に関係するレジスタ

ADPCM(MSM6258V)

アドレス	READ/WRITE	bit 7	6	5	4	3	2	1	bit 0	備　　考
$E92001	R	PLAY/REC	'1'	'0'						ADPCMステータス
	W	'0'					REC ST	PLAY ST	SP	〃　コマンド
$E92003	R/W	$Data_{n+1}$				$Data_n$				データ入出力

PPI(i8255)ポートC/コントロールワードレジスタ

アドレス	READ/WRITE	bit 7	6	5	4	3	2	1	bit 0	備　　考
$E9A005	R/W	IOA6	IOA5	PC5	PC4	Sampling RATE		PCM	PAN	ADPCMサンプルレート/出力制御
$E9A007	W	'0'				Bit Sel			Data	ポートCのビット単位での制御

OPM(YM2151)レジスタNo.=$1B($E90001に$1Bを書き込んでからアクセスする)

アドレス	READ/WRITE	bit 7	6	5	4	3	2	1	bit 0	備　　考
$E90003	W	CT2	CT1					W		ADPCM基本クロック切り替え

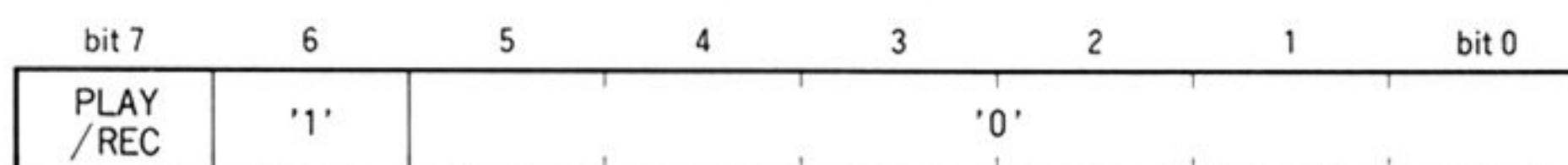
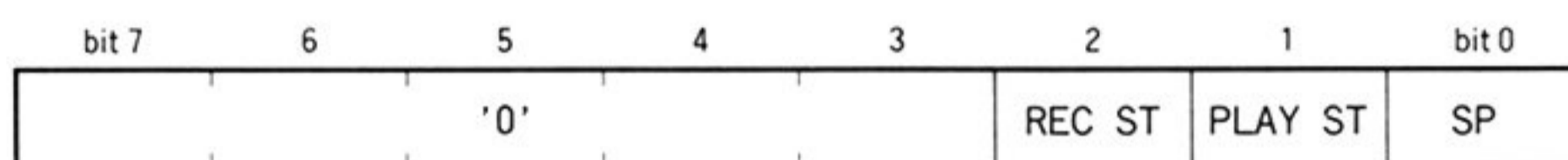
なお，ADPCMのデータ転送はDMACで行うほうが便利なので，X 68000ではDMACのチャンネル3をADPCM用に割り付けています。DMACの設定方法などについてはDMAの章を参照してください。

❸・❷1　ADPCMステータスレジスタ

ADPCMステータスレジスタのビット配置を294ページの図33に示します。ビット7は，ADPCMが録音（音声データ入力）ないしスタンバイ中であるか，再生（音声データ出力）中であるのかを示すビットです。'1'のとき録音中/スタンバイ状態，'0'のときは再生中であることを示しています。

❸・❷2　ADPCMコマンドレジスタ

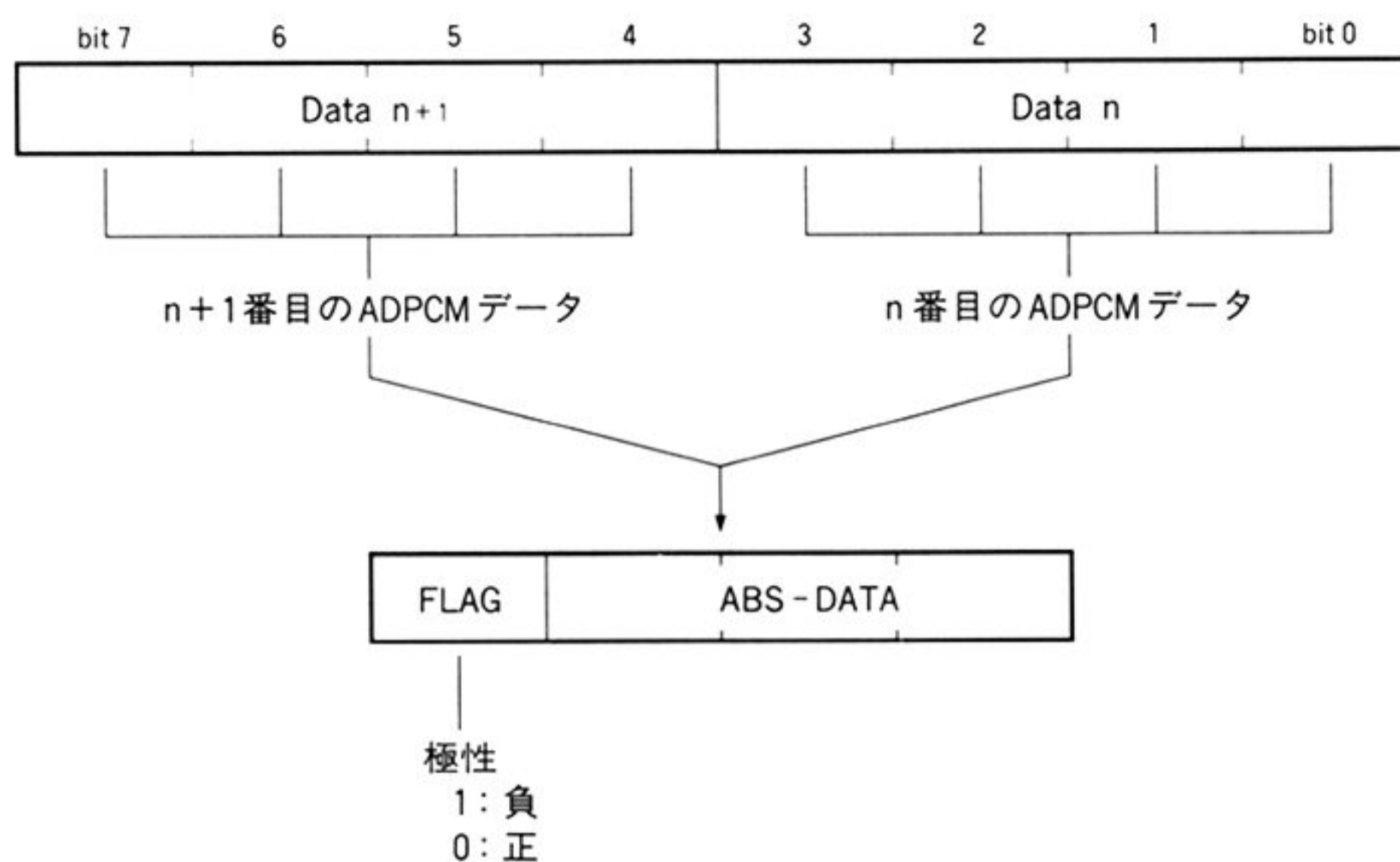
ADPCMの動作開始/停止制御を行います。ビット配置は294ページの図34のようになっています。ビット2が録音開始，ビット1が再生開始の制御を行い，ビット0は録音/再生動作の停止を指示するビットです。再生動作が終了したときはコマンドレジスタで停止を指示しないと，ADPCMは最後に与えたデータを繰り返し使用して音声出力を行ってしまいますので，

●図……33　ADPCM ステータスレジスタ（$E92001）

●図……34　ADPCM コマンドレジスタ（$E92001）

必ず停止コマンドを書き込むようにしてください。

また，停止コマンドを書き込んだとき，ADPCM の出力レベルは最後の状態のまま保持され，次の再生開始コマンドを受け取ったときに 1/2 VDD（最大振幅の半分）に戻されるため，最後の出力レベルが 1/2 VDD 近辺でないと，開始コマンドを与えた直後に「ボツッ」という音が出ることがあります。

❸·❷3 ADPCMデータレジスタ

ADPCM データの入出力を行うレジスタです。ビット配置は図 35 のようになります。ADPCM データは 2 サンプリング分ずつまとめて転送を行いますので，図のように，上位 4 ビットと下位 4 ビットに分かれており，下位 4 ビットが先，上位 4 ビットが後のサンプリングで作成されたデータになっています。それぞれの 4 ビットデータは最上位ビットが符号，下位 3 ビットが絶対値となっています。

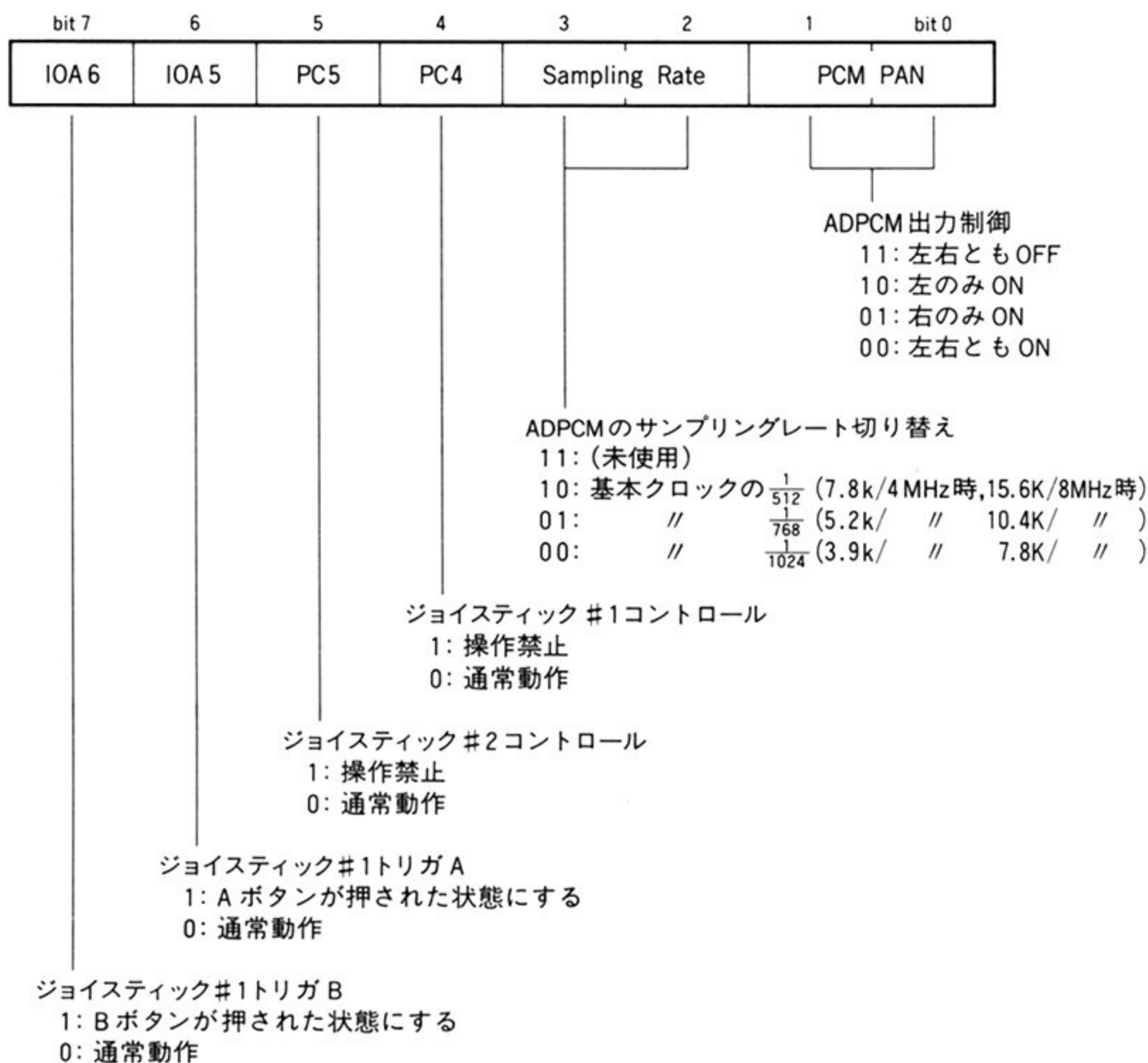
●図……35　ADPCMデータレジスタ（$E92003）

❸·❷ 4 PPI(8255)ポートC

ジョイスティック用に使用しているPPIの空きビットを使用して，サンプリングレートの選択やADPCMのパンポット制御に使用しています。このポートのビット配置を296ページの図36に示します。

ビット3とビット2は，ADPCMのサンプリングレートを基本クロックの1/512，1/768，1/1024のいずれにするかを選択します。2ビットが'11'のパターンは未使用扱いになっています。実際に設定してみると，'01'のときと同じ結果になるようです。

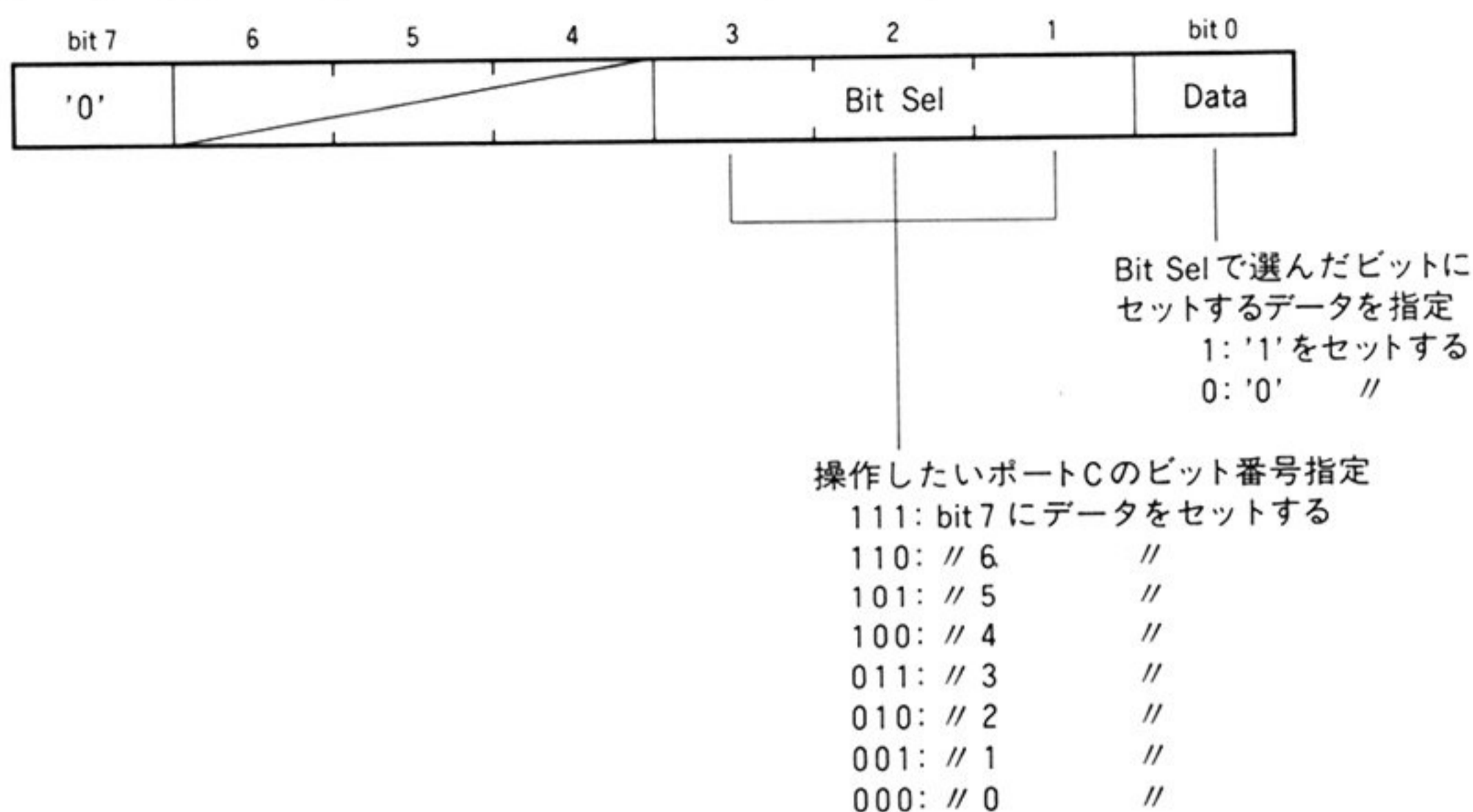
ビット1とビット0はパンポット制御で，ビット0が左チャンネル，ビット1が右チャンネルの制御用となっており，それぞれ'1'にすると出力がOFF，'0'にするとONになります。片チャンネルだけONにすればそちらから，両方ともONにすると中央から音が出ているように聞こえます。

なお，X68000では8255のポートCを出力ポートとして使用していますが，8255の特性上，出力ポートを読み出すと，出力されているデータがそのまま読み出せますので，現在設定されているデータを読み出して必要なビットだけを変更することができます。

●図……36　8255 ポートC（$E9A005）

❸・❷5 PPI (8255) コントロールワードレジスタ

PPIは，特殊機能として，ポートCの任意のビットを操作するビットセット/リセット機能を持っています。この操作はPPIのコントロールワードレジスタで行います。ビットセット/リセットコマンドのビット配置を図37に示します。ビット7が'0'であるとき，8255はビットセット/リセットコマンドと認識し，ビット1からビット3を操作したいポートCのビット位置，ビット0をセットしたいデータとみなして，指定されたビットだけを与えられたデータに変更します。

むろん，ポートCはリード/ライト可能ですから，いったんセットされているデータを読み出してからANDやORなどのビット演算を行い，再度書き込んでもかまわないのですが，操作

●図……37　8255コントロールワードレジスタ（$E9A007）

するビットが1つだけであるような場合にはビットセット/リセット機能を使うほうがかんたんだと思われるので，この機能を説明しておきました。

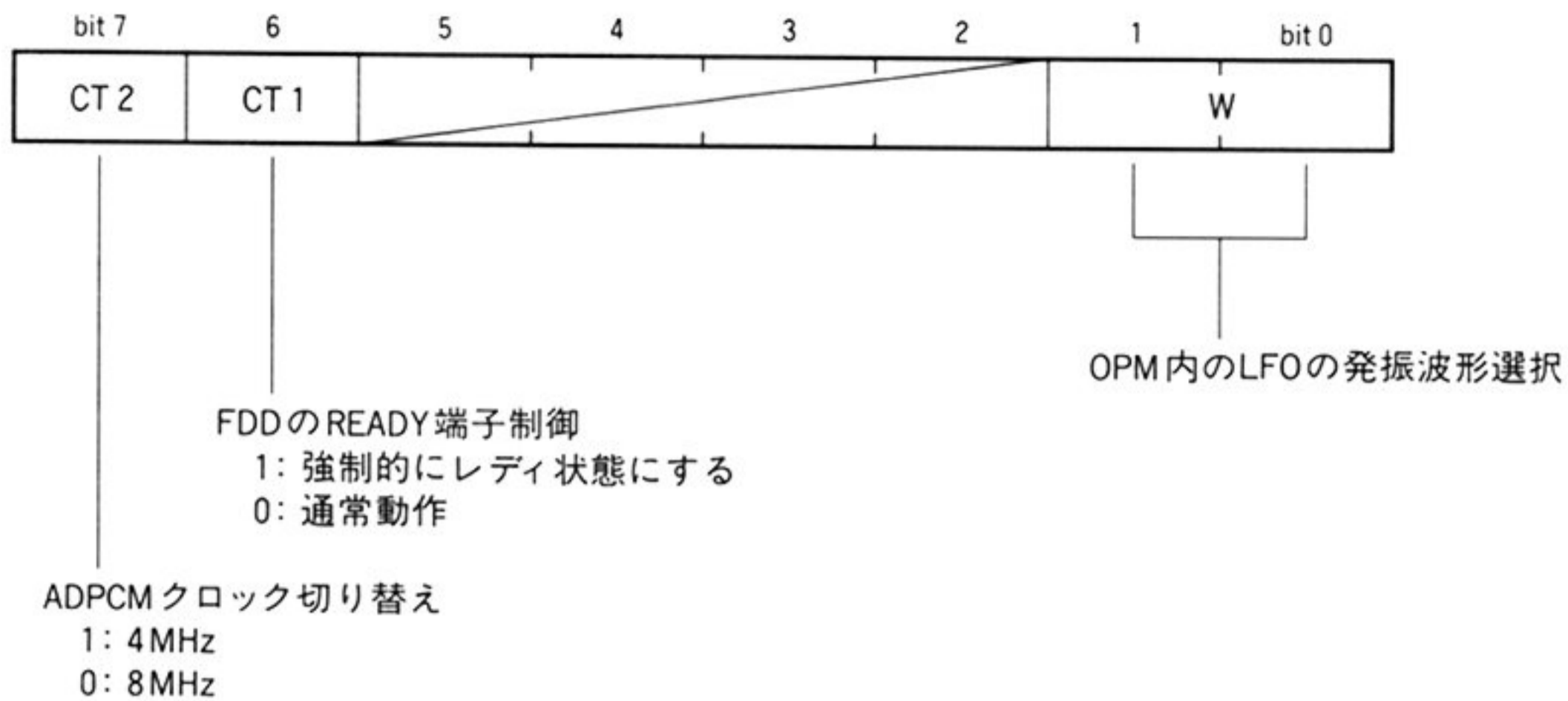
❸·❷6　OPMレジスタ$1B

X 68000では，ADPCMのクロックの切り替え信号としてOPM（FM音源LSI）の汎用出力端子CT 2を使用しています。このビットはOPM内の番号$1 Bのレジスタで行います。レジスタのビット配置は298ページの図38のようになっています。このレジスタのビット7がクロック選択用のビットで，'0'のとき8 MHz，'1'のとき4 MHzになります。

このレジスタは書き込み専用で，読み出すことができないため，システム的に使用する場合には，ほかのビットの値に気を配る必要があります。

❸·3　サンプルプログラム

ADPCMの操作を行うサンプルプログラムを作成してみましたので参考にしてください。この例では，$1Fの連続データの再生を行わせています。起動時のオプション指定で，第1引き数がPPIのパンポット制御ビット（ビット0，1）にセットする値，第2引き数がPPIのサンプリングレート選択ビット（ビット2，3），第3引き数がADPCMの基本クロック選択（OPM

●図……38　OPM レジスタ（レジスタ No.＝$1B）

のレジスタ$1B）に設定する値です。

このサンプルでは PPI の制御にビットセット/リセット機能を使ってみましたので，あわせて参考にしてください。

●リスト……1　ADPCM の操作（$1F の連続データの再生）

```
/*
 * ＡＤＰＣＭ動作テスト
 *
 * XC ではvolatile がサポートされていないため、
 * 次の１行を入れてvolatileを無効にしてください
 * #define  volatile
 */

#include <doslib.h>

struct  DMAREG {
    unsigned char   csr;
    unsigned char   cer;
    unsigned short  spare1;
    unsigned char   dcr;
    unsigned char   ocr;
    unsigned char   scr;
    unsigned char   ccr;
    unsigned short  spare2;
    unsigned short  mtc;
```

```
    unsigned char   *mar;
    unsigned long   spare3;
    unsigned char   *dar;
    unsigned short  spare4;
    unsigned short  btc;
    unsigned char   *bar;
    unsigned long   spare5;
    unsigned char   spare6;
    unsigned char   niv;
    unsigned char   spare7;
    unsigned char   eiv;
    unsigned char   spare8;
    unsigned char   mfc;
    unsigned short  spare9;
    unsigned char   spare10;
    unsigned char   cpr;
    unsigned short  spare11;
    unsigned char   spare12;
    unsigned char   dfc;
    unsigned long   spare13;
    unsigned short  spare14;
    unsigned char   spare15;
    unsigned char   bfc;
    unsigned long   spare16;
    unsigned char   spare17;
    unsigned char   gcr;
} ;

volatile struct DMAREG  *dma;
volatile unsigned char  *ppi_cwr;   /* 8255コントロールワードレジスタ*/
volatile unsigned char  *opm_regno; /* OPMレジスタ番号設定レジスタ   */
volatile unsigned char  *opm_data;  /* OPMデータレジスタ           */
volatile unsigned char  *adpcm_command; /* ADPCMコマンドレジスタ     */
volatile unsigned char  *adpcm_status;  /* ADOPCMステータスレジスタ  */
volatile unsigned char  *adpcm_data;    /* ADPCMデータレジスタ       */

#define BUFSIZE 0x400
unsigned char   pcmbuf[BUFSIZE];

void main();
void create_adpcmdata();
void adpcm_outsel();
```

```
void adpcm_sample();
void adpcm_clksel();
void adpcm_stop();
void adpcm_start();
void dma_setup();
void dma_start();
void wait_complete();
void clear_flag();

void main(argc,argv)
    int argc;
    char    *argv[];
{
    unsigned int    i,pan,sample,clk;
    if (argc >= 2)
        pan = atoi(argv[1]);
    else    pan = 0;
    if (argc >= 3)
        sample = atoi(argv[2]);
    else    sample = 0;
    if (argc >= 4)
        clk = atoi(argv[3]);
    else    clk = 0;
    SUPER(0);
    dma           = (struct DMAREG *)0xe840c0;
    ppi_cwr       = (unsigned char *)0xe9a007;
    opm_regno     = (unsigned char *)0xe90001;
    opm_data      = (unsigned char *)0xe90003;
    adpcm_command = (unsigned char *)0xe92001;
    adpcm_status  = (unsigned char *)0xe92001;
    adpcm_data    = (unsigned char *)0xe92003;

    adpcm_stop();
    create_adpcmdata(pcmbuf,BUFSIZE);
    adpcm_outsel(pan);       /* Panpot Control   */
    adpcm_sample(sample);        /* Sampling rate    */
    adpcm_clksel(clk);       /* ADPCM Clock      */
    clear_flag();
    dma_setup();
    dma_start();
    adpcm_start();
```

```
    wait_complete();
    adpcm_stop();
    clear_flag();
}

void create_adpcmdata(buf, length)
    unsigned char   *buf;
    unsigned int    length;
{
    while(length--)
        *buf++ = 0x1f;
}

void adpcm_outsel(sel)
    unsigned int    sel;
{
    *ppi_cwr = (0 << 1) | ((sel >> 1) & 1); /* Left     */
    *ppi_cwr = (1 << 1) | (sel & 1);    /* Right    */
}
void adpcm_sample(rate)
    unsigned int    rate;
{
    *ppi_cwr = (2 << 1) | ((rate >> 1) & 1);
    *ppi_cwr = (3 << 1) | (rate & 1);
}

void adpcm_clksel(sel)
    unsigned int    sel;
{
    *opm_regno = 0x1b;
    *opm_data  = (sel & 1) << 7;
}

void adpcm_stop()
{
    *adpcm_command = 0x1;
}

void adpcm_start()
{
    *adpcm_command = 0x2;
}
```

```
void dma_setup()
{
    dma->dcr = 0x80;
    dma->ocr = 0x32;
    dma->scr = 0x04;
    dma->ccr = 0x00;
    dma->cpr = 0x08;
    dma->mfc = 0x05;
    dma->dfc = 0x05;

    dma->mtc = BUFSIZE;
    dma->mar = pcmbuf;
    dma->dar = (unsigned char *)adpcm_data;
}

void dma_start()
{
    dma->ccr |= 0x80;
}

void wait_complete()
{
    while(!(dma->csr & 0x90) && !(*adpcm_status & 0x80))

}

void clear_flag()
{
    dma->csr = 0xff;
}
```

❸·4 ADPCMデータ

ADPCM データへの変換アルゴリズムが具体的にどのようになっているかについて調べたのですが，メーカ（沖電気）のノウハウに該当するものであることから非公開ということでした。このため，X 68000 の ADPCM データがどのようになっているかは不明です。

ADPCMについて調べているときに見つけたヤマハの音源LSI, YM2608（OPNA）に内蔵されているADPCM音源のアルゴリズムをコラムに載せておきますので参考にしてください。

C O L U M N

ADPCMのアルゴリズム（ADPCM音声分析の手順）

① A/D変換 …音声をサンプリングレートごとに8 bitのPCMデータに変換します

② 8→16 ……得られたPCMデータを256倍して16 bitのデータ；Xnに変換します

③ dnの算出 …このXnを予備値xnと比較して，その差分；dnを求めます

④ ADPCMデータの決定

………dnが正のときはADPCMのデータのMSB（L 4）を'0'，負のときは'1'にします

差分の絶対値；｜dn｜と量子化幅；⊿nの関係から，ADPCMデータの残り3 bit（L 3，L 2，L 1）を決定します

ADPCMデータの符号化は表Aに示すとおりです

●表……A　ADPCMデータと量子化変化率（f）

L4		L3	L2	L1	f	条件 (1n=1 dn 1/Δn)
dn≧0	dn＜0					
0	1	0	0	0	57/64	In＜1/4
		0	0	1	57/64	1/4≦In＜1/2
		0	1	0	57/64	1/2≦In＜3/4
		0	1	1	57/64	3/4≦In＜1
		1	0	0	77/64	1≦In＜5/4
		1	0	1	102/64	5/4≦In＜3/2
		1	1	0	128/64	3/2≦In＜7/4
		1	1	1	153/64	7/4≦In

以上の操作で，音声データからADPCMデータへの変換は終わりです

⑤ 予測値と量子化幅の更新

………ADPCMデータが得られると，次ステップの予測値；xn+1と量子化幅；⊿n+1の更新を行います

$$Xn+1=(1-2\times L4)\times(L3+L2/2+L1/4+1/8)\times \varDelta n+xn$$

$$\varDelta n+1=f(L3,\ L2,\ L1)\times \varDelta n \quad :\varDelta nmin=127,\ \varDelta nmax=24576$$

＊初期設定：予 測 値　X1=0
量子化幅　⊿1=127

以下，①～⑤の操作を各サンプリングタイムごとに繰り返して音声分析が行われます

SCC

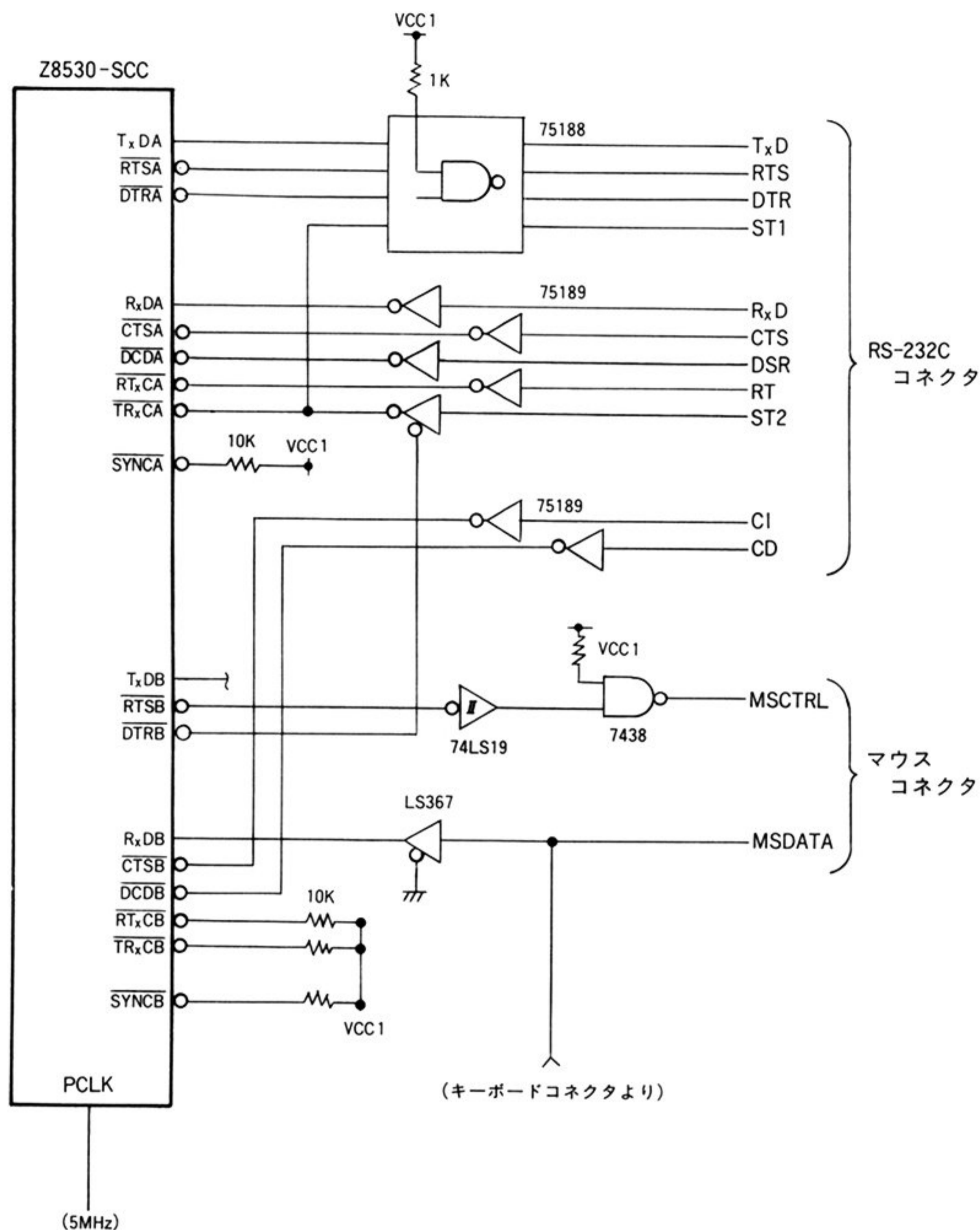
SCCは，シリアル伝送LSIで，同期通信，非同期通信のほか，データの変復調までサポートしています。ここでは，X 68000でのSCCの使われ方や，SCCの各レジスタの設定法などについて説明します。

1 SCCの概要

X 68000では，RS-232 Cインタフェースとマウスをサポートするための LSI として，Z 8000のファミリーLSIであるZ 8530 SCC（シリアルコミュニケーションコントローラ；以下，たんにSCCと略します）を使用しています。

X 68000のSCC周辺のブロック図を306ページの図1に示します。

SCCは，チャンネルAとチャンネルBの2つのシリアルポートを持っているのですが，X 68000では，このうち，チャンネルBのRTSとRxDをマウス用に使用し，チャンネルAをRS-232 Cポートに利用しています。また，SCCの持っている信号線は，RS-232 Cをサポートするには少々不足しているため，チャンネルBのうち，使われていないCTS，DCDをそれぞれCIとCDライン入力に，DTRをクロック切り替えに流用しています。これによってX 68000では，一般的な非同期通信だけでなく，Monosync や Bisync，SDLC といった同期通信も標準でサポートできるようになっています。また，SCCはデータに変調をかけたり，変調された信号からデータとクロックを分離する（復調）機能が内蔵されており，2線式の同期通信などが容易に行えるようになっています。

●図……1　SCC 周辺ブロック図

X 68000 のハード上使用可能な通信モードを次に示します。ただし，X 68000 では，SCC とのデータ転送に DMA が使用できないため，転送速度は CPU の応答速度に依存します。このため，実際に利用可能な転送速度は，ここに掲げた数値よりかなり落ちると思われます。

チャンネルA

・非同期モード時

キャラクタ長　　：5/6/7/8 ビット
ストップビット長：1/1.5/2 ビット
パリティ　　　　：偶数/奇数/なし
クロック　　　　：×1, ×16, ×32, ×64（×1 は外部で同期をとる必要あり）
エラー検出　　　：パリティエラー
　　　　　　　　　オーバーランエラー
　　　　　　　　　フレーミングエラー

・同期モード時

バイト指向同期モード（Monosync, Bisync）

キャラクタ同期　　　：内部/外部いずれも可
同期キャラクタ数　　：1/2 個
同期キャラクタ長　　：6/8 ビット
同期キャラクタ制御：自動挿入/削除可
CRC　　　　　　　　：自動生成/チェック可

SDLC モード

アボートシーケンス自動生成/検出
自動ゼロ挿入/削除
メッセージ間フラグ自動挿入
アドレスフィールド自動検出
Information フィールドの端数処理
CRC 自動生成/チェック
SDLC ループモード時の EOP 検出による自動オンループ/オフループ

・データ転送速度

非同期モード　　　　　：38.4 Kbps（×16 モード時）
Monosync/Bisync　　　：1.5 Mbps
FM 符号化方式 DPLL　 ：375 Kbps
NRZI 符号化方式 DPLL ：187 Kbps

チャンネルB

非同期通信のみ

キャラクタ長　：8ビット

ストップビット：2ビット

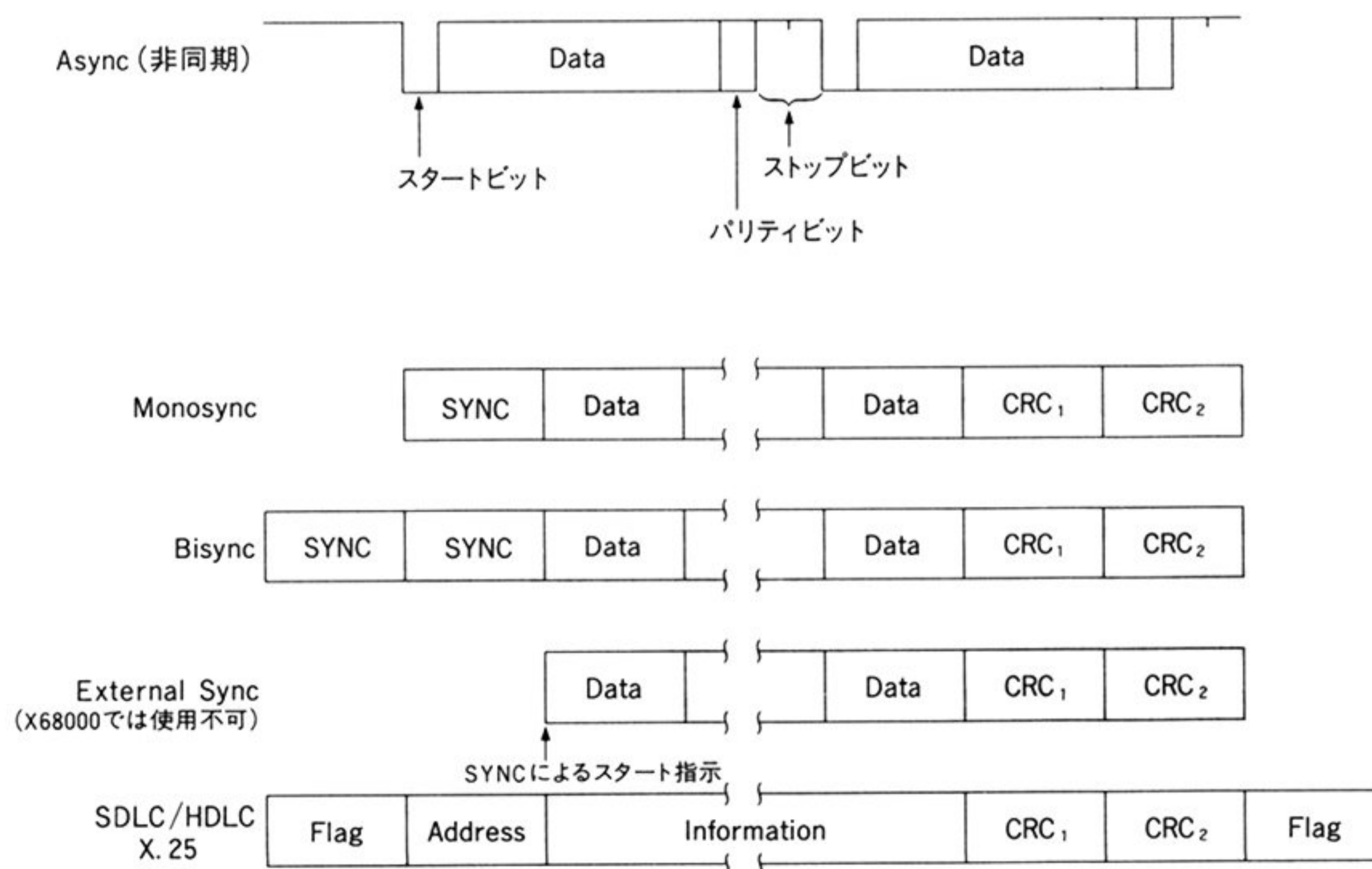
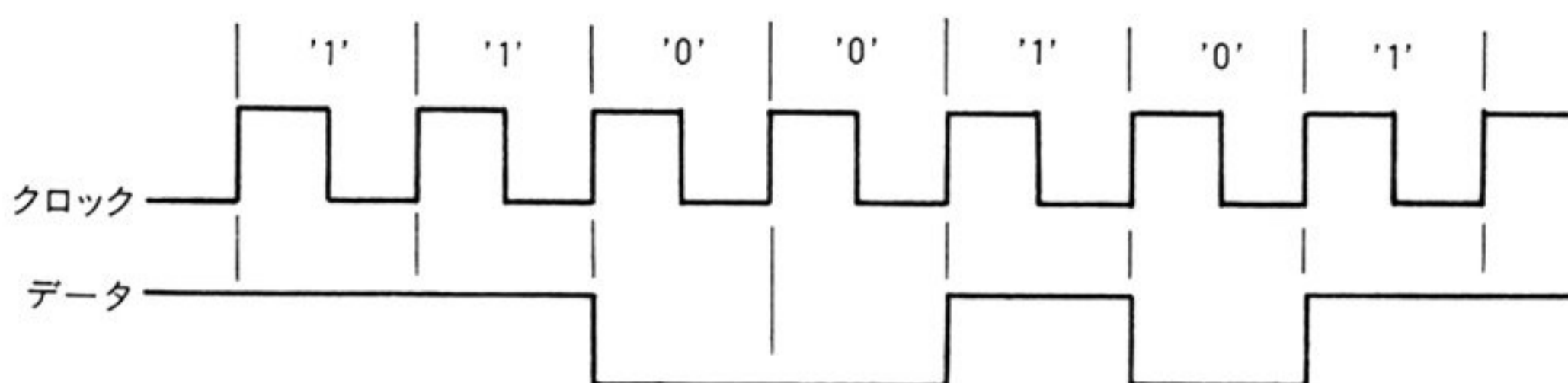
パリティ　　　：なし

ボーレート　　：4800 bps

❶·1 SCCのデータ通信モード

SCCのサポートする通信モードとそのデータフォーマットを図2にまとめてみました。Async(非同期)モードは，現在，もっとも一般的に使用されているもので，俗にRS-232Cサポートというときには，このデータフォーマットでのデータ伝送がサポートされていることを指します。

そのほかのフォーマットはいずれも同期伝送モードです。同期伝送モードは，いずれもデータのほかに送受信タイミングをとるためにクロック信号が必要となります。クロックとデータの関係の例を図3に示します。送信側は，クロックの立ち上がりに同期してデータを変化させ，

●図……2　SCCがサポートするデータフォーマット

●図…… 3　同期伝送の動作

受信側はクロックの立ち下がりのタイミングデータを取り込むようにすることで，確実にデータが受け渡せるようにするわけです。

データの前にある Sync や Flag は一連のデータの先頭であることを示すとともに，受信側が確実にデータとタイミングをあわせるために使用されるものです。データの後に続く CRC は，受け取ったデータにエラーがないかどうかを検出するためのチェックコードです。

これらのうち，External Sync モードは，データの読み始めのタイミングを LSI の SYNC 端子を使って伝えることになっているのですが，RS-232 C の信号には該当するものがないこともあって，X 68000 では LSI の SYNC 端子を外部に引き出していません。このため，X 68000 では，このモードを使用することはできません。

次に，これらのデータフォーマットについてもう少し詳しく見ていくことにしましょう。

❶･❶1　Async（非同期）モード

図 2 では非同期通信モードの典型的なデータフォーマットを示しました。データの前には 1 ビットのスタートビットと呼ばれる'0'のビットがまず送られ，データの後にはチェック用のパリティビット，さらに 1 キャラクタ分のデータの最後を示すストップビットと呼ばれる'1'のデータが送信されます。パリティビットは省略される場合もあります。また，ストップビットのビット長は，1 ビット，1.5 ビット，2 ビットのいずれかが選択可能です。

受信側では，データラインが'0'になったのを見つけると，データの取り込み準備を開始し，各ビットのほぼ中央（と思われるところ）を順次サンプリングしていきます。データに続くパリティビットはデータの正確性をチェックするための 1 ビットのデータです。パリティには，データとパリティビットを含めた'1'の総数が偶数になるようにパリティビットのデータを決める偶数パリティと，奇数になるようにする奇数パリティがあります。さらに，受信側はデータの最後を示すストップビットがあるのを確認します。もし，ストップビットがあるはずのところで'0'が読み出された場合にはエラー（フレーミングエラー）となります。

❶·❶ 2 Monosync（モノシンク）モード

Monosync モードにかぎらず，同期通信モードでは，送受信ともクロック信号に同期してデータの出力/入力を行います。このため，データのほかにクロック信号が必要となりますが，Async モード時に1キャラクタごとに付加されてしまうスタートビットやストップビットといった余分なデータが不要であることから，効率のよい伝送が行えます。

フレーム（一連のデータ列）の最初には SYNC（同期）キャラクタと呼ばれる同期用のデータが，最後には CRC コードによるチェックデータが付加されます（SYNC キャラクタデータは SCC の WR 7 レジスタにセットした値が使用されます）。

同期通信モードでは，Async モードのような，1キャラクタの先頭や最後を示すデータパタンは存在しないため，どのビットがデータの最初であるのか見分けがつきません。このため，SYNC キャラクタという特別なデータパタンを用意しておき，これをフレームの先頭として認識するようにするわけです。

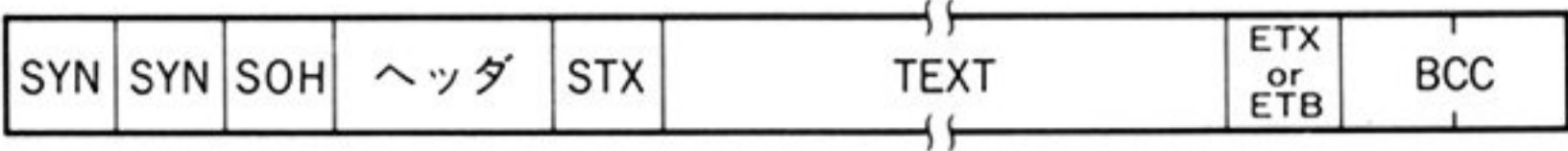
SCC で Monosync モードを使用するときに，データを読み始めるときやデータを取りそこなった場合など，同期をとり直す必要が生じた場合にはレシーバをハントモードにします。SCC は，このモードに設定されると，SYNC キャラクタパタンと同一のビットパタンが見つかるまで待ち続け，一致したパタンが受信できた時点から順次データの取り込みを開始します。

❶·❶ 3 Bisync（バイシンク）モード

Bisync（Binary Synchronous Communication）は IBM が提唱した通信方式で，メッセージのフォーマットは SYNC キャラクタが2つになったほかは Monosync とよく似ています。図4に Bisync メッセージのフォーマット例を示します。最後の BCC は Block Check Code の略で，先ほど掲げた図では CRC として示されるものです。SOH や STX などは，制御コードとして割り当てられているデータを指します。これらの制御コードの一覧を図5に示しておきます。

Bisync では（Monosync でも同じ），SYN などのデータ制御コードを特別なデータとして扱うため，テキストの中にこれらのデータが入り込むと，それをテキストとして受け取ればよいのか，制御コードとして処理すべきなのか区別できなくなります。つまり，このままのフォーマットではバイナリデータ（音声，画像データ，実行可能ファイルなど）の送受信は行えないことになります。

●図……4　Bisync メッセージフォーマット

SYN	SYN	SOH	ヘッダ	STX	TEXT	ETX or ETB	BCC

●図……5　伝送制御キャラクタ例

符号	値	名称	意味
SOH	$01	Start Of Heading	ヘディング開始
STX	$02	Start of TeXt	テキスト開始
ETX	$03	End of TeXt	テキスト終結
EOT	$04	End Of Transmission	伝送終了
ENQ	$05	ENQuiry	問い合わせ(相手局からの応答要求)
ACK	$06	ACKnowledge	肯定応答
DLE	$10	Data Link Escape	伝送制御拡張
NAK	$15	Negative AcKnowledge	否定応答
SYN	$16	SYNchronous Idle	同期信号
ETB	$17	End of Transmission Block	伝送ブロック終結

IBMでは，これに対処する方法として，制御データの前にDLE($10)を挿入する方法をとりました。このモードは透過伝送モードと呼ばれます（バイナリデータの伝送に対応していないほうは通常伝送と呼んでいるようです)。このモードでは，たとえば，SYN ($16)はDLE SYN ($1016) という2バイトデータになって送られます。$10というデータを送りたいときにはDLE $10 ($1010) という2バイトデータに変換することで対処します。透過モードで付加/削除されるDLEコードはCRCの計算には含めないことになっています。

SCCのBisyncモードは透過モードをサポートしていませんので，透過モードを使うときにはCPUでDLEの挿入/削除やCRC計算などの処理を行う必要があります。

❶·❶4 External Sync(外部同期)モード

外部同期モードでは，データの開始位置をMonosyncやBisyncのような特殊データで認識するのではなく，ハード的に(SYNC端子で) 制御してもらうようになっています。すでに

述べたように，X 68000ではSCCのSYNC端子を使用していませんので，このモードでの伝送は行えません。

❶·❶5 SDLCモード

Monosync や Bisync によって伝送できるデータがあくまでもバイト単位であるのに対して，SDLC はビット単位での伝送を考慮したモードであり，任意のビット数の情報の伝送が行えるようになっています。

図6にSDLCメッセージフォーマットを示します。Bisyncなどでは，DLEやSYNなどを特殊なデータとして扱っていましたが，ビット単位の伝送が基本であるSDLCでは，そのような方法はとれないため，'1'が6つ連続するデータである'01111110'の8ビットデータをフレームの先頭と終了を示すフラグデータとして用い，それ以外のところでは，'1'が5つ連続すると，'0'を挿入するようにしています。

受信側では，'1'が5つ続いた後に'0'がきた場合にはその'0'を削除し，'1'が6つ連続してきた場合にはフラグとみなすことで，データを誤ってフラグだと思い込んだりする恐れがなくなります。

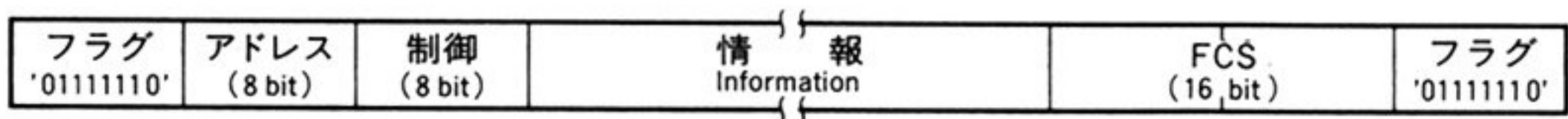
アドレスフィールドは，送信側が指定した受信相手の番号を示すものです。SDLCは，1対1のデータ伝送だけでなく，多くのステーションが同一の伝送ラインを使用するネットワーク環境を想定しています。このような環境では，伝送するネットワーク上の，どのステーションにデータを送りたいのかを明示する必要があります。SDLCでは，各ステーションに8ビットの番号（アドレス）を振り，送信したフレームがどこあてのものであるのかを明示しているわけです。アドレスの値のうち$FFはグローバルアドレスと呼ばれ，不特定の相手にコマンドを送るために使用します。

制御フィールドは転送フレームの番号やフレームの種別を識別するためのデータで，データ長は8ビット固定になっています。

インフォーメーションフィールドには，実際に送受信するデータが入っています。この内容はなんら規定はなく，総ビット数も任意でかまいません。

インフォーメーションフレームの後に付加されるFCS (Frame Check Seaquence) は16ビットのCRCデータで，フレームの内容が正しく受信されたかどうかをチェックするための

●図……6　SDLCメッセージフォーマット

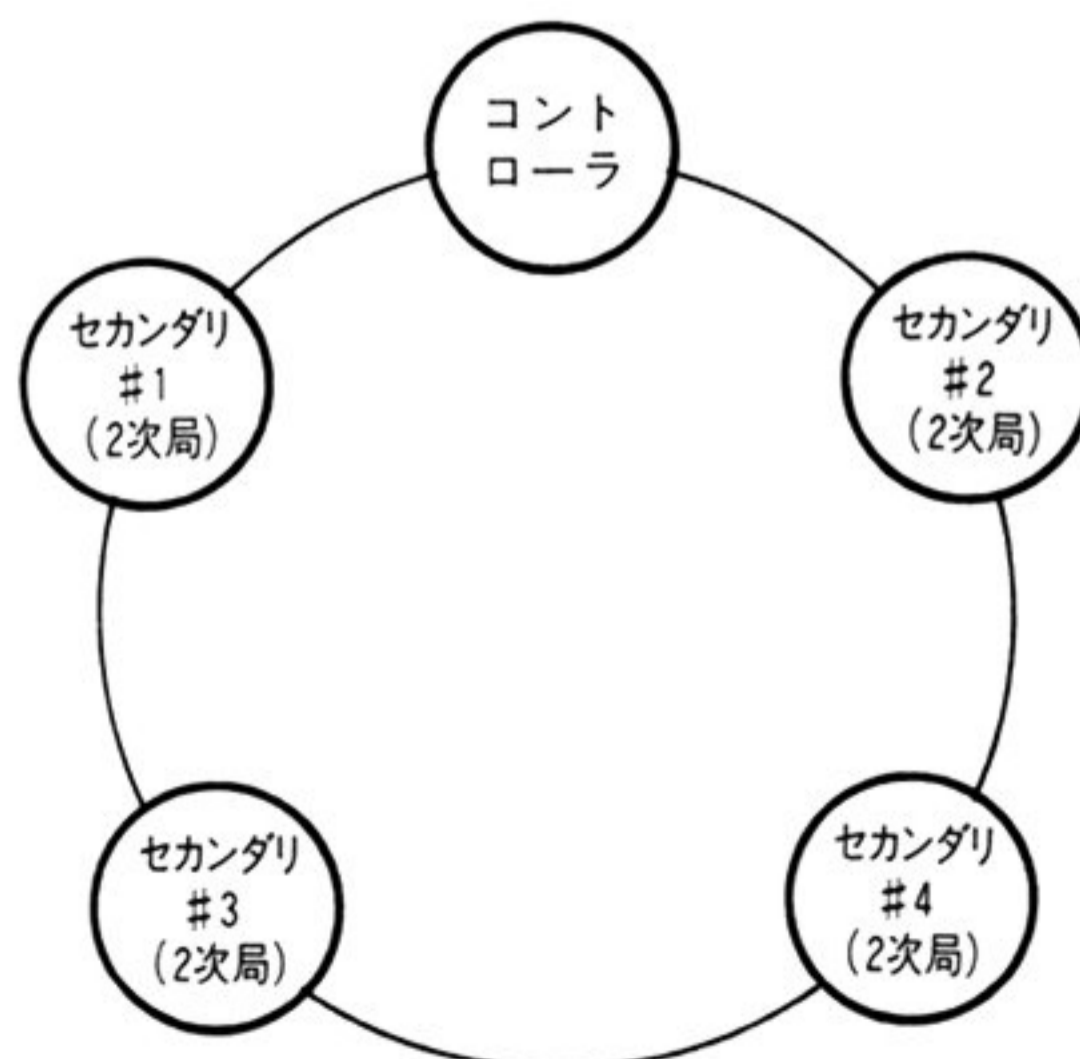
フラグ '01111110'	アドレス (8 bit)	制御 (8 bit)	情報 Information	FCS (16 bit)	フラグ '01111110'

ものです。CRCの計算方法は何種類もあるのですが, SDLCではCRC-CCITT方式と呼ばれる方法をとっています。SCCでは, WR5のビット2を'0'にすることで, CRC-CCITT方式でCRCの生成/チェックが選択されます。

❶·❶6 SDLCループモード

SDLCループモードは, 通常のSDLCを少し拡張して, 1つの親局(コントローラ：1次局)の下に多数の子局（セカンダリ：2次局）が接続され, 1次局がループ上のすべてのデータ伝送を制御するような用途に適するようにしたものです。SDLCループモードのシステムの構成例を図7に示します。

SDLCループモードでは, メッセージは一度に全局に伝えられるのではなく, ループ上の各局を巡回していきます。2次局は送られてきたメッセージを受信しつつ, 次の局にメッセージを渡していきます。

2次局によるメッセージの送信はいつでも行えるわけではなく, 1次局からのメッセージ送出許可があったときに限られます。1次局は, 2次局がメッセージを送出してもかまわないときには, EOP (End Of Poll) と呼ばれる特殊なデータ（'11111110'）を送出します（SDLCと同様, SDLCループモードでも, 通常のデータでは'1'が5つ続くと'0'を自動的に挿入しますから, EOPデータがデータ中に偶然検出される恐れはありません)。2次局はEOPを受け取ったとき, もし送出したいメッセージがあったなら, EOPの最後の'0'を'1'に変更して送出した後に自分の送出したいメッセージを送出し, 最後にEOPを付加します。何も送出したいメッセー

●図……7　SDLCループ

ジがない場合は通常どおり，ただ受け取ったメッセージを隣りに送り直すだけです。SDLCループモードでは，データにNRZIやFM変調をかけることもできます。

❶·2 ボーレートジェネレータ

SCCは，内部にボーレートジェネレータと呼ばれる，伝送のための基準クロック作成回路があり，任意の伝送速度を選択することができるようになっています。ボーレートジェネレータには16ビットのレジスタがあります。このレジスタへの設定値Nと，ボーレートジェネレータから出力される周波数fの関係は，SCCのPCLK端子に与えられている周波数（X68000では5MHz）をPCとすると，f=PC/(2×(N+2))となります。

この出力周波数が実際の伝送レートと一致するのは，同期通信でデータの変調機能を使わないときだけで，そのほかの場合にはタイミングをとったり，変調のかかったデータからデータとクロックへの分離を行ったりするために，伝送速度(単位：bps)の16倍や32倍程度の高い周波数が必要となりますので，ボーレートジェネレータからの出力周波数もそれを考慮して決める必要があります。

たとえば，一般的な非同期モードでは，データがどのタイミングで入ってくるかわからないため，SCCは実際の伝送速度よりも速いクロックでデータラインをサンプリングし，データビットのほぼ中央でデータを読み込むようにしています。SCCでは非同期モードでのサンプリングクロック周波数と伝送速度の比を16，32，64から選択できるようになっています。

図8に×16モードで非同期通信を行うときの伝送速度とボーレートジェネレータの設定値の関係をまとめておきましたので参考にしてください。X68000では，クロックが5MHzと半端な値であるために公称伝送速度と実際の伝送速度には若干のずれが出ていますが，非同期通信では1キャラクタごとに同期をとり直しているようなものなので，2パーセント以下のずれであれば，まず問題になることはありません。

❶·3 データの符号化

SCCは，4種類の符号化手法を選択することができるようになっています。それぞれの符号化によるデータラインの動きの違いを図9に示します。

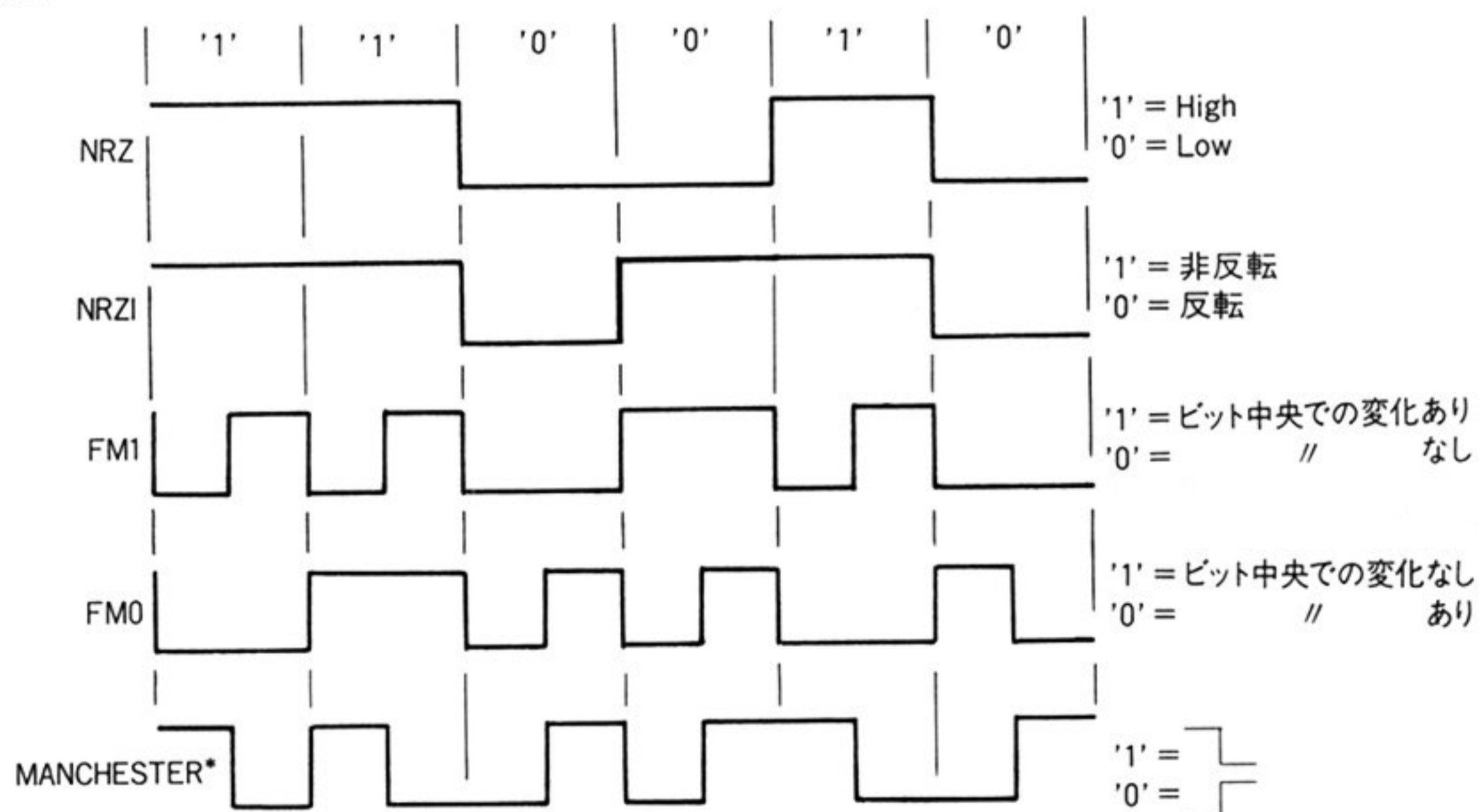
NRZは，データの'1'，'0'がそのまま出力の'1'，'0'に対応するもので，もっとも一般的に使用されているものです。

NRZIは，データが'0'のときに出力が反転し，'1'のときは反転しないというもの，FM1は

●図……8　ボーレートジェネレータへの設定値（参考）

公称伝送速度 (bps)	ボーレートジェネレータへの設定値	実際の伝送速度 (bps)	公称値との比
38400	2 ($0002)	39062.5	1.017
19200	6 ($0006)	19531.3	1.017
9600	14 ($000E)	9765.6	1.017
4800	31 ($001F)	4734.8	0.986
2400	63 ($003F)	2403.8	1.002
1200	128 ($0080)	1201.9	1.002
600	258 ($0102)	601.0	1.002
300	519 ($0207)	299.9	1.000
150	1040 ($0410)	150.0	1.000
75	2081 ($0821)	75.0	1.000

＊クロックモード×16のときの値

●図……9　SCCがサポートする符号化モード

＊MANCHESTERは，DPLLをFM，レシーバをNRZモードにすると復号できる

データが'0'のときにはビットの境界で，'1'のときにはビットの境界と中央で出力を反転させる手法，FM 0 は FM 1 とは逆に'0'のときにビットの中央で反転させるようにしたものです。NRZ 以外の符号化データを受信するときは，入力信号からデータを取り出す（復号）ために次に述べる DPLL 回路を使用されます。

また，特殊なモードとして，SCC のレシーバを NRZI モード（WR 10 のビット 5，6 を利用），DPLL を FM モード（WR 14 のビット 5，6，7 を利用）にすることでマンチェスター符号を受信することができます（送信は不可）。

NRZ 以外の符号化データを受信する場合には，DPLL によって入力信号からデータに同期したクロックを作成し，データとクロックの分離を行うことができますので，伝送速度がある程度わかっていれば，外部にクロック分離回路など，復号化のための特別な回路を付加する必要はありません。

❶·4　DPLL

SCC は内部に DPLL（Digital Phase Locked Loop）回路を内蔵しており，特別な外部回路を付加せずに，NRZI や FM 変調のかかったデータからクロックとデータを分離することができるようになっています。DPLL の基本クロックは，NRZI 変調データを扱うときには伝送速度（単位：bps）の 32 倍，FM 変調データの場合には 16 倍の周波数が必要となります。

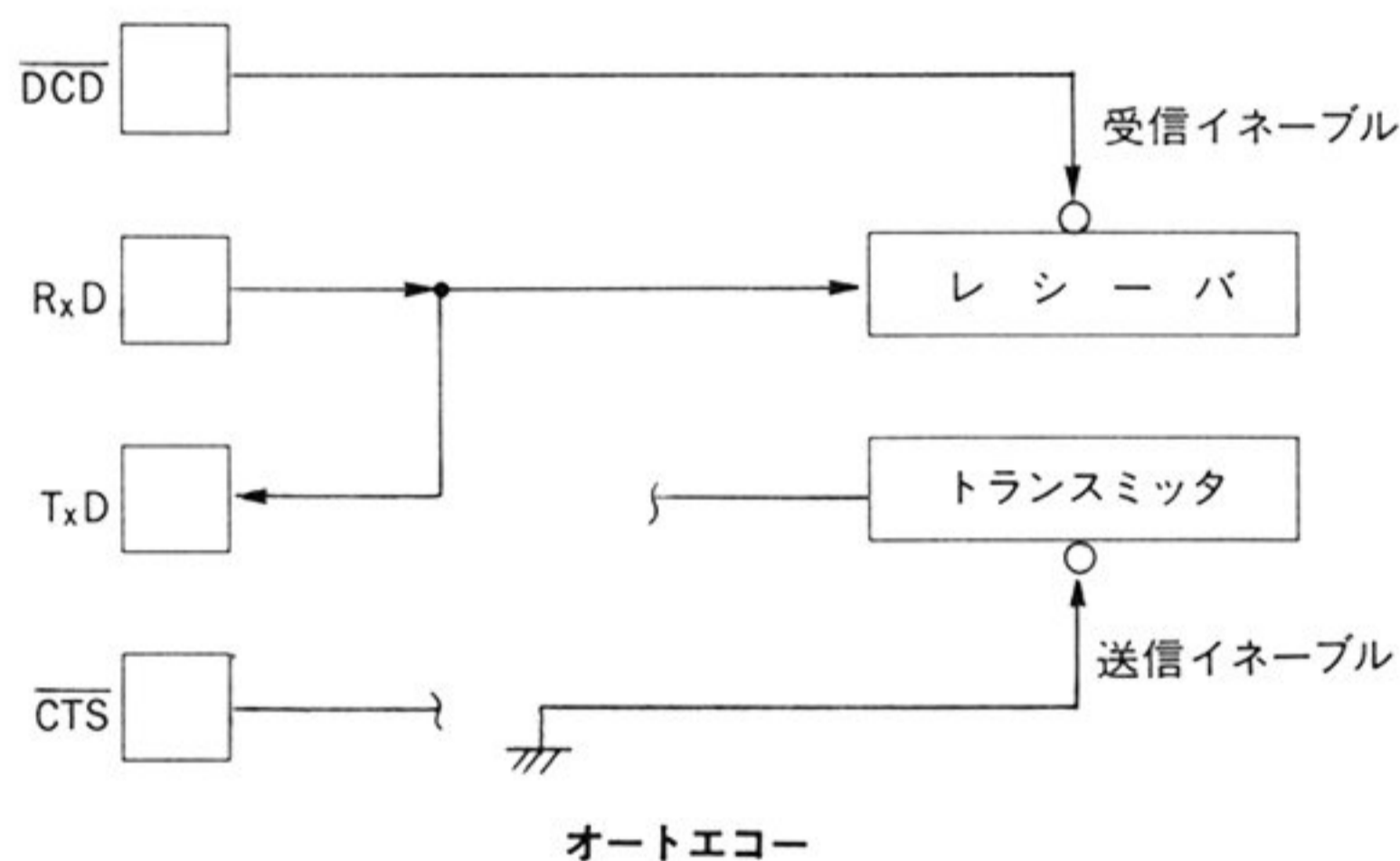
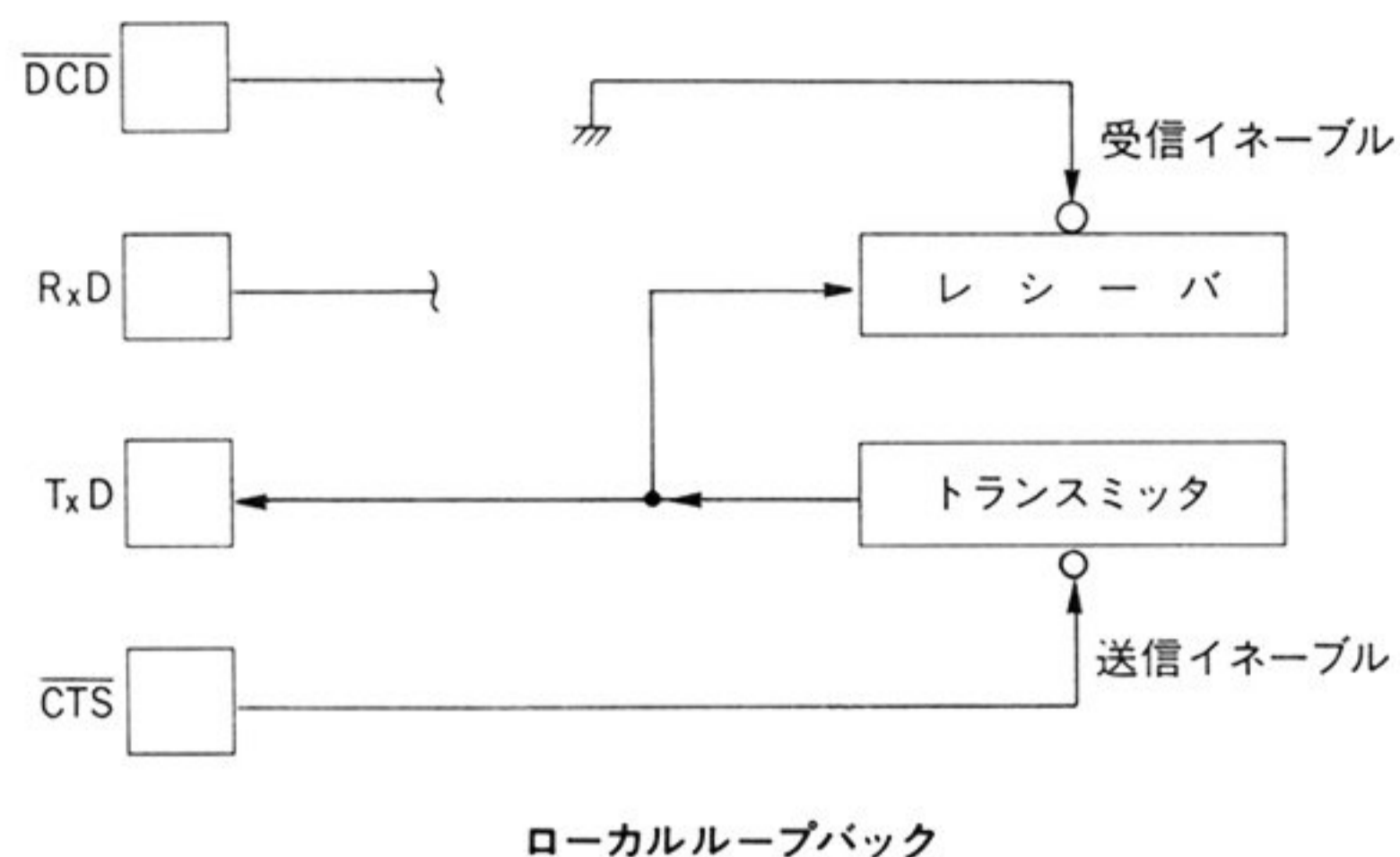
DPLL は，入力波形の変化をとらえて，自分のサンプリングクロックが正しいタイミングになっているかどうかを常時チェックしており，ずれが生じたときにはサンプリングクロックを調整してデータの伝送速度に同期するようにしています。これによって，送信側と受信側での伝送速度のずれなどがあっても正しくデータが取り出せるようになっています。

❶·5　ローカルループバックとオートエコー機能

ローカルループバックは送信したものがそのまま自分でも受信される動作，オートエコーは受信されたデータを自動的に送信する動作のことです。それぞれの動作の概要を図 10 に示します。これらのモードは非同期，SYNC，SDLC のいずれのモードでも使用することができます。

ローカルループバックでは，レシーバはトランスミッタと直結され，たとえオートモードにプログラムしていても，DCD 端子はレシーバの動作制御信号としては動作しません。

●図……10　ローカルループバックとオートエコー

ローカルループバック

オートエコー

オートエコーでは，RxD から受信されたものがそのまま TxD に出力されます。トランスミッタの出力は切り離されてしまいますので，オートモードであっても，CTS 端子はトランスミッタの動作制御信号としては動作しなくなります。

❶·6 割り込み

SCC の割り込み発生要因は，各チャンネルごとに 4 種類ずつ持っており，それぞれの要因ごとに異なる割り込みベクタを生成できます。Human 68 K は，割り込み要因によって，ベクタのビット 1 から 3 までが変化するモードで使用しており，発生するベクタ番号は$50 から$5 E

までとなっています。各チャンネルの割り込みの種類は次の4種類です。

スペシャルRxコンディション

受信データにパリティエラーがあったり，オーバーランが発生してしまったような場合に発生します。

受信キャラクタ有効

データが受信され，受信バッファに有効なデータが入ったときに発生します。割り込みで受信を行う場合には，この割り込みで受信バッファからデータを引き取るようにします。

送信バッファ空

送信バッファに空きができたことを示します。割り込みを受けたら，次のデータを送信バッファに書き込むようにします。

E/S（外部/ステータス）変化

RS-232 Cの制御線の変化やSCC内部で発生したステータス（送信アンダーランなど）など，他の3種の割り込みにいずれにも該当しない割り込み要因は，すべてこの割り込みになります。この割り込みが発生したときはRR 0で具体的な要因を知ることができます。

❶·7 SCCのレジスタ

X 68000におけるSCCのポートアドレスを図11に，レジスタの一覧を図12に示します。SCCは，独立した2つのチャンネルを持っているため，ポートアドレスもそれぞれ専用に持っています。このうち，コマンドポート($E98001，$E98005)はSCC内部のレジスタのアクセスに，データポート($E98003，$E98007)は実際に伝送を行うデータの入出力を行うためのポートです。

SCC内部には，図12に示すように書き込み用のレジスタが16本，読み出し用のレジスタが

●図……11 SCCポートアドレス

アドレス	bit 7	6	5	4	3	2	1	bit 0	備考
$E98001									チャンネルBコマンドポート
$E98003									チャンネルBデータポート
$E98005									チャンネルAコマンドポート
$E98007									チャンネルAデータポート

●図……12　SCCのレジスター覧

レジスタ	レジスタの機能
WR 0	CRCの初期化，SCCの初期化コマンド，アクセスするレジスタの選択
WR 1	送受信の割り込みの設定
WR 2	割り込みベクタの設定
WR 3	受信動作パラメータの設定
WR 4	送受信動作に関係するパラメータの設定
WR 5	送信動作パラメータの設定
WR 6	同期キャラクタ/SDLCのアドレス設定
WR 7	〃　/SDLCのフラグ設定
WR 8	送信バッファ（$E98007(チャンネルA)，$E98003(チャンネルB)と同一）
WR 9	CPUへの割り込み発生制御，SCCのリセット
WR10	トランスミッタ/レシーバの各種制御
WR11	クロックモード制御
WR12 WR13	ボーレートジェネレータ（R13：上位，R12：下位）
WR14	DPLL動作モード等の設定
WR15	外部/ステータス割り込み発生許可/禁止制御
RR 0	送受信バッファや制御端子ステータス
RR 1	スペシャルRxコンディションステータス，端数コード等の読み出し
RR 2	割り込みベクタ（チャンネルA：WR2への設定値，チャンネルB：最後に発生した割り込みベクタ番号）
RR 3	ペンディングされている割り込み要因の読み出し（チャンネルA側のみ存在する）
RR 8	受信バッファ（$E98007(チャンネルA)，$E98003(チャンネルB)と同一）
RR10	FMモードでのMissing Clock，SDLCでの動作ステータス等
RR12 RR13	ボーレートジェネレータへの設定値（WR12/WR13への設定値）
RR15	WR15に設定した値が読み出される

9本ありますが，これらのレジスタをアクセスするのにコマンドポート1つだけですませられるようにSCCでは少々変わった方法を使用しています。

SCCは，通常レジスタ番号0のレジスタ（WR 0やRR 0）がアクセスできるようになっていて，これ以外のレジスタをアクセスするときにはWR 0にレジスタ番号を書き込み，続いて，そのレジスタにセットたい値を書き込みます。書き込みが終わると，ふたたびWR 0やRR 0がアクセスされるようになります。

このような方法をとっていると，プログラムミスなどで書き込み回数などがずれると，WR 0にレジスタ番号を書き込んでいるつもりが，他のレジスタへ書き込んでしまったりすることになります。このような心配があるときには，SCCのコマンドポートにダミーのリードを行えばよいのです。もし，レジスタ番号を設定した後なら，このリードによってレジスタ番号0がアクセスされるようになりますし，すでにレジスタ番号0になっているなら，RR 0の内容が読み出されるだけでSCCの動作にはなんの影響もありません。

レジスタ番号8のレジスタ（WR 8とRR 8）へのアクセスは，データポートへのアクセスと同じです。直接アクセスが可能であるのに，わざわざレジスタ番号8を指定してから，読み書きするなどという手間のかかることをする必要があるとは思えませんが，SCCにはこのようなアクセス方法もサポートされています（WR 8とRR 8がデータレジスタであることからすると，書き込み用レジスタは15本，読み出し用は8本というほうが正確かもしれません）。

次に，それぞれのレジスタの内容について，もう少し詳しく見ていくことにしましょう。

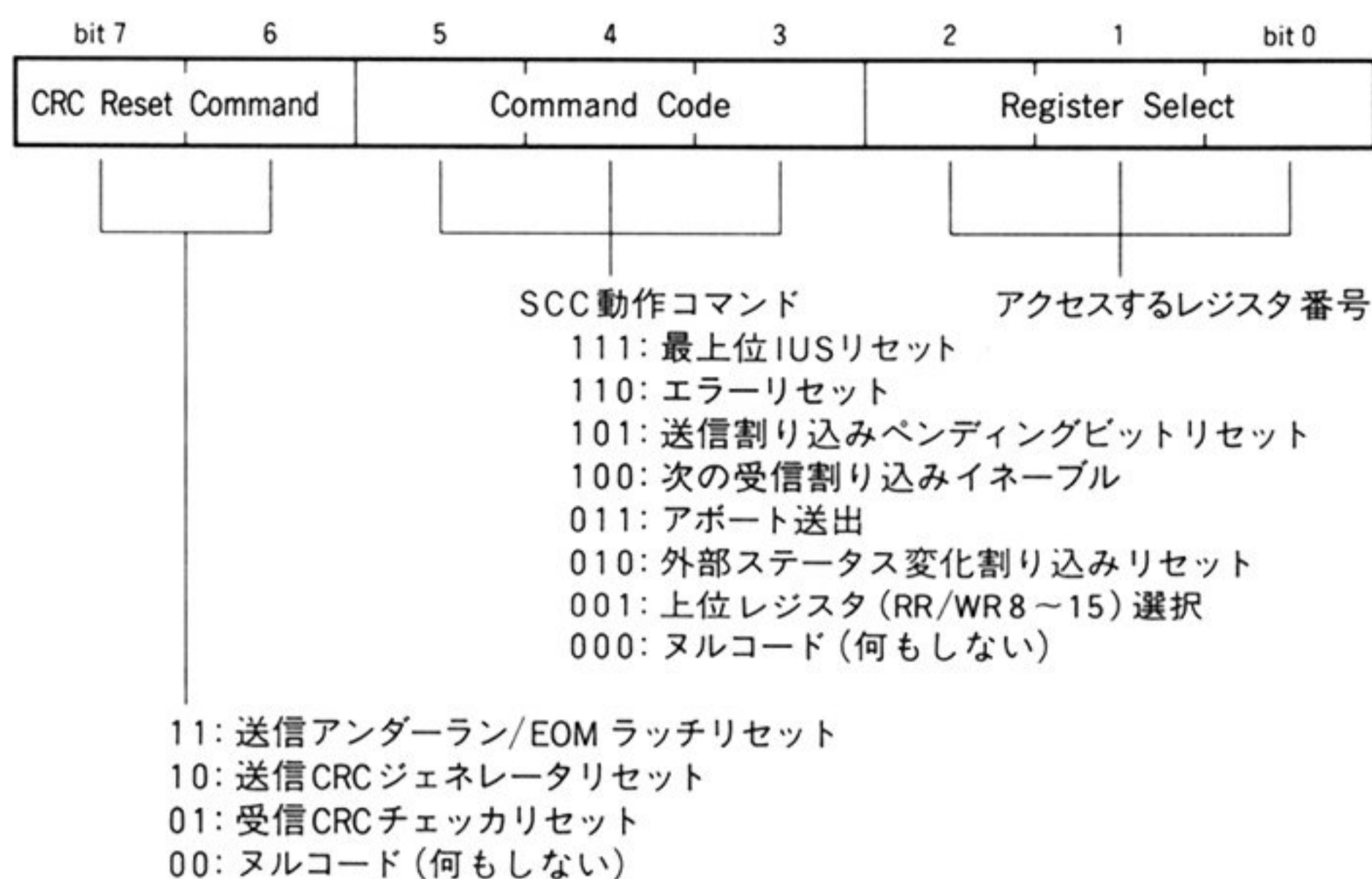
❶・❼1 WR 0

WR 0のビット配置を図13に示します。それぞれのビットの内容の詳細は，次のようになっています。

1 ビット7, 6（CRC Reset Command）

CRCチェッカ/ジェネレータの制御を行うのに使用します。CRCジェネレータは，SCCのチャンネルリセットコマンドを発行しても初期化されませんので，必ずこのコマンドで初期化する必要があります。

送信アンダーラン/EOM（送信終了）ラッチコマンドは，CRC送出制御のために使用されます。RR 0のビット6が'0'のときに送信アンダーラン/EOMが発生した（つまり，すべてのデータを送り終わったと見なされる）場合，SCCはCRCデータを送信し，RRのビット6を'1'にセットします。これをクリアするのが送信アンダーラン/EOMラッチコマンドです。

●図……13　WR 0

2 ビット5, 4, 3（Command　Code）

WR 0のビット5，4，3の3ビットはSCCへのコマンドコードです。それぞれのコマンドの意味は，次のようになっています。

111（最上位 IUS リセット）

サービス中の割り込みのうち最上位のものをクリアし，下位の割り込み要求を可能にします。割り込み処理の最後では必ずこのコマンドを発行するようにしないと，次の割り込みが入ってこれなくなります。

110（エラーリセット）

スペシャル Rx コンディション割り込み（要因は RR 1の上位4ビットで読み出されます）をクリアします。スペシャル Rx コンディション割り込みが発生した場合，SCC は，このコマンドが発行された時点で割り込み要因となったデータをバッファから削除します。つまり，スペシャル Rx コンディション割り込みが発生した時点のデータが必要な場合は，バッファを読み出した後でこのコマンドを発行することになります。

101（送信割り込みペンディングビットリセット）

送信割り込み（'送信バッファ空'の割り込み）が発生したとき，それ以上送信するものがない場合にはこのコマンドを発行して，それ以上送信割り込みが発生しないようにします。このコマンドを発行しないまま，最上位 IUS リセットコマンドを発行すると，発行したとたんにふたたび送信割り込みが発生してしまいます。

100（次の受信割り込みイネーブル）

WR 1のビット3，4を'01'（'最初のキャラクタで割り込み発生'）に設定したとき，最初の受信処理の中でこのコマンドを発行すると，次のデータが受信されたときにも割り込みが発生するようになります。すでに受信バッファにデータが入っているときも，このコマンドを発行すると，受信割り込みが発生します。

011（アボート送出）

SDLCモードで，アボート（8～13個の連続した'1'）を送出するために使用します。このコマンドを発行すると，送信バッファは自動的に空となり，RR 0のビット6（送信アンダーラン/EOM）が'1'になります。

010（外部ステータス変化割り込みリセット）

E/S（外部/ステータス変化）割り込みが発生した場合，割り込み処理の中でこのコマンドを発行しておきます。E/S割り込みの要因は，RR 0の該当ビットが'1'になっていることで判断できます。このコマンドを発行することで，このステータスが'0'にクリアされます。

001（上位レジスタ選択）

WR 0の下位3ビットはSCCの内部レジスタの選択に使用されますが，レジスタ番号が8以上のものを選択するときにはビット5，4，3を'001'にします。ビット7，6は通常'00'（ヌルコード）としますので，WR 0にたんにレジスタ番号を書き込めば，レジスタ番号が8以上のときはCommand Codeビットは自然に'001'（つまり，このコマンド）になります。

000（ヌルコード）

SCCの動作には影響ありません。レジスタ番号が0～7を選択するときには，これらのビットは自然に'000'となります。

3 ビット2，1，0（Register Select）

次のリード/ライト動作で，SCC内部のレジスタの中からどれにアクセスするかを指定します。この値が'000'～'111'でレジスタ番号0～7を示します。レジスタ番号が8以降のレジスタの場合には，Command Codeを'001'にすれば，このビットによって8～15が選択されます。

❶・❼ 2 WR 1

WR 1のビット配置は図14のようになっています。WR 1は，送受信割り込みの禁止/許可などの制御，データ転送モードの設定などを行います。

●図……14　WR 1

1 ビット7, 6, 5(W/REQ信号制御)

これらのビットは，SCCの持っているW/REQ信号の動作を制御するものです。W/REQ信号はSCCがデータ転送の準備ができたことを示す信号で，アクセスされたときにデータ転送準備ができるまでDMAやCPUを待たせるウェイト信号や，DMACへの転送要求信号として用いられるものです。X 68000では，この信号線は接続されていません。

ビット7は，W/REQ信号の機能を使用するか否かを選択するビットで，このビットを'1'にするとW/REQ信号が有効になります。X 68000ではW/REQ信号は使用されていませんので，このビットは'0'に設定します。

ビット6は，W/REQ端子をウェイト信号として機能させるか，DMA転送要求信号として動作させるかを選択するもので，'1'にするとDMA動作，'0'でWait信号として動作するようになります。

ビット5は，W/REQ信号を受信動作のときに使用するか，送信動作のときに使用するかを選択するものです。'1'にすると受信動作，'0'にすると送信動作に対応して動作するようになり

ます。

2 ビット4, 3（受信割り込みモード）

受信割り込みをどのような条件で発生させるかを指定するものです。

11（スペシャルRxコンディション時のみ割り込み）

スペシャル Rx コンディションが発生したときに受信割り込み発生とするモードです。このモードのとき，割り込みが発生した時点で割り込み要因となったデータは，WR 0 にエラーリセットコマンドを発行するまで受信バッファに残ったままとなっています。このモードはDMA 転送を利用するときには便利なモードなのですが，X 68000 では，SCC は DMAC には接続されていないため，あまり利用されることはないでしょう。

10（すべての受信キャラクタで割り込み）

X 68000 では，通常このモードを使うことになるでしょう。データが1つ受け取られるごとに CPU に対して受信割り込みを発生します。スペシャル Rx コンディション条件が成立すれば，スペシャル Rx コンディション割り込みが発生します。スペシャル Rx コンディションとなった要因は，RR 1 の上位 4 ビットに示されています。

01（最初の受信キャラクタで割り込み）

このモードでは，受信動作を開始して以降，最初に受信されたキャラクタで割り込みを発生します。スペシャル Rx コンディション条件が成立すれば，スペシャル Rx コンディション割り込みが発生します。

00（受信割り込み禁止）

このモードでは受信割り込みの発生が禁止されます。CPU への割り込み出力が禁止されるだけで，RR 0 のステータスビットは割り込み発生時と同様に動作し，チャンネルBの RR 2 の割り込みベクタも生成されますので，CPU は RR 0 や RR 2 をチェックしながら受信動作を行うことができます。

3 ビット2（パリティエラーをスペシャルRxコンディション割り込みとする）

パリティエラーをスペシャル Rx コンディション割り込みとするか否かを選択できます。'1'のとき，パリティエラーはスペシャル Rx コンディション割り込み要因になります。

4 ビット1（送信割り込み制御）

送信割り込みを発生させるか否かを制御します。このビットを'1'にすると送信割り込みが許可，'0'で禁止となります。

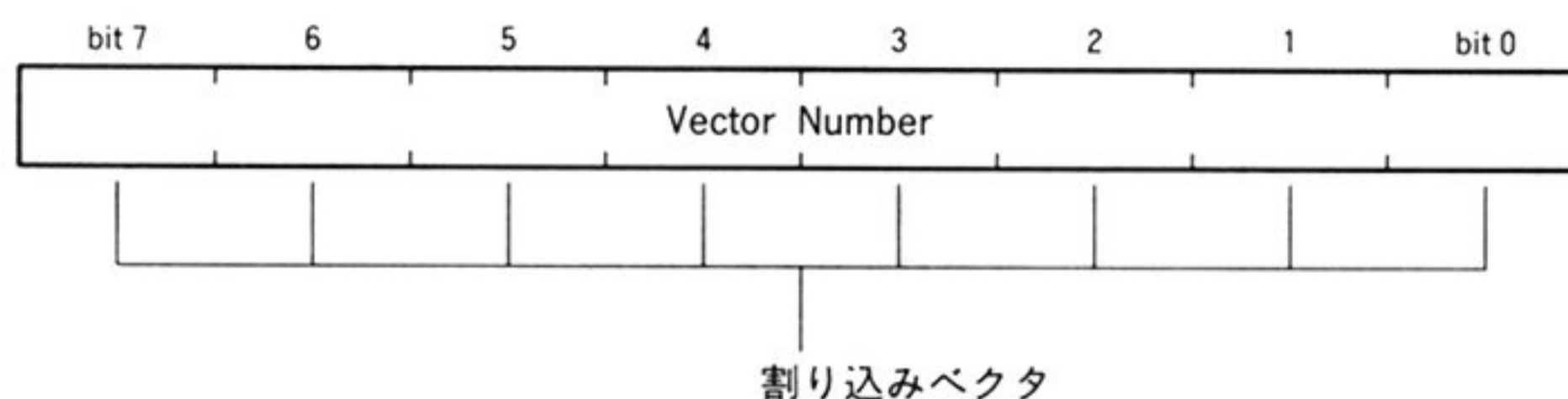
5 ビット0（E/S割り込み制御）

E/S（外部/ステータス変化）割り込みは，RR 0のビット2とビット0以外のいずれかの条件が成立したときに発生します。このビットは，このE/S割り込みの許可/禁止を制御するもので，'1'のときE/S割り込み発生が許可，'0'で禁止となります。

❶・❼ 3 WR 2

WR 2のビット配置を図15に示します。このレジスタは，SCCが発生する割り込みベクタを設定するものです。SCCは，WR 9のビット0が'1'のとき，割り込み要因に応じて各チャンネルごとに4つ，計8種の割り込みベクタを生成します。このとき，割り込み要因によって，ビット4～6を変化させるか，ビット3～1を変化させるかをWR 9のビット4で選択します。

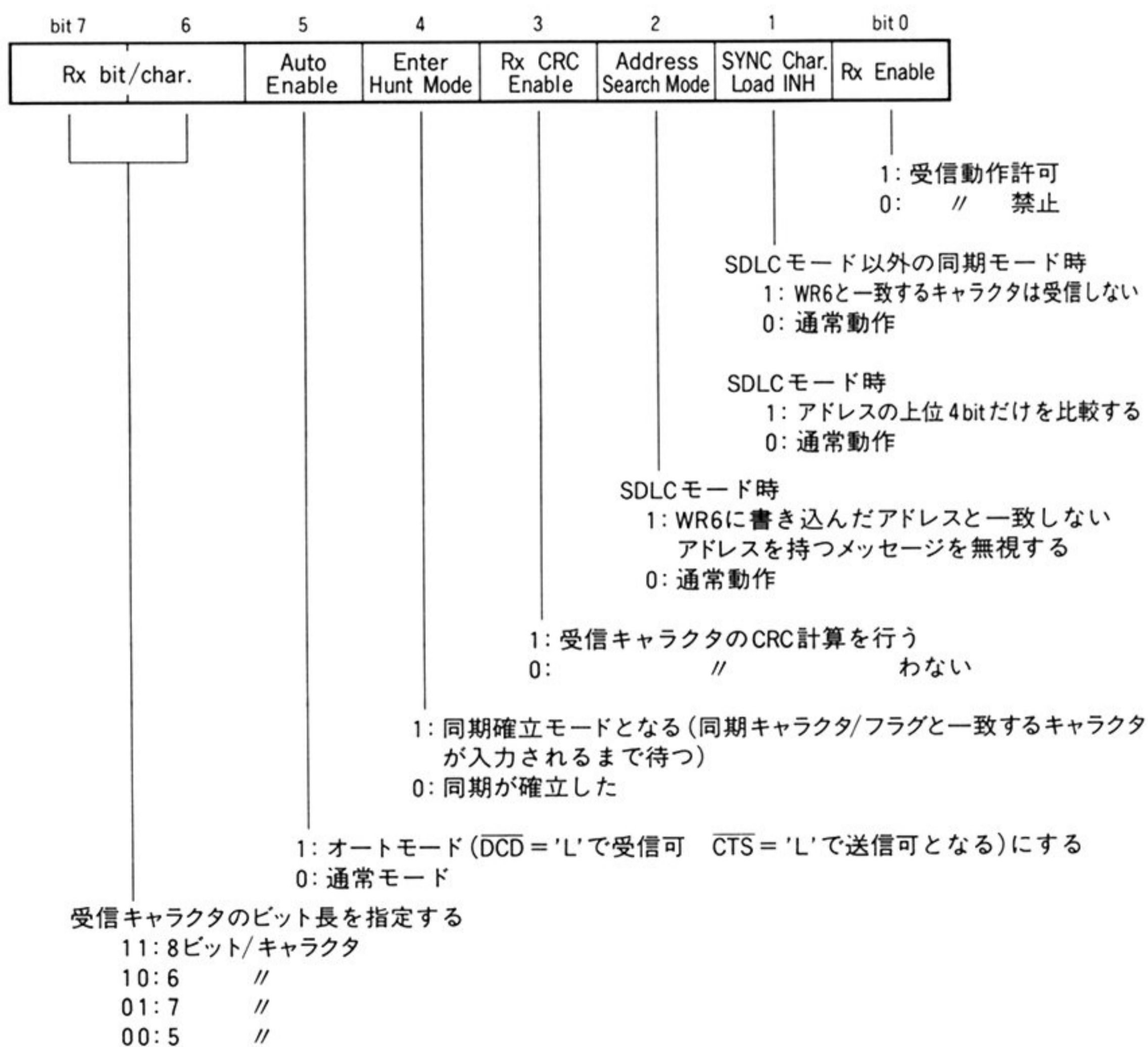
Human 68 KではWR 2に$50を，WR 9のビット4と0をそれぞれ'0'，'1'として，割り込み要因によってベクタのビット3～1が変化するモードで使用しています。これによりSCCは，割り込み要因によって，$50，$52，$54，$56，$58，$5A，$5C，$5Eの8種類のうち，いずれかを発生することになります。

●図……15　WR 2

❶·❼4 WR 3

WR 3のビット配置を図16に示します。このレジスタは，受信キャラクタのビット長など受信動作の制御を行うものです。それぞれのビットの意味は次のようになっています。

1 ビット7,6(受信キャラクタのビット長)

非同期モードでの受信時の1キャラクタあたりのビット長を指定します。SYNCモード(Monosync, Bisync)やSDLCモードでは，つねに8ビット単位で受信されます。受信キャラクタ長を8ビット以下にした場合には，余った上位ビットはすべて'1'になります。

●図……16 WR 3

2 ビット5（オートイネーブル）

このビットを'1'にすると，トランスミッタやレシーバの動作がCTSやDCD信号で制御されるようになります。オートモードでは，CTS端子が'Low'レベルになると送信動作が許可になり，DCD端子が'Low'レベルになると受信動作が行われるようになります。つまり，CTSが送信許可信号，DCDが受信許可信号として機能するわけです。

すでに述べたように，オートモードにプログラムしても，ローカルループバックモードでのDCD端子，オートエコーモードでのCTS端子は制御端子としては動作しなくなります。

3 ビット4（エンターハントモード）

このビットを'1'にすると，SCCは受信データと同期をとり直すモードになり，WR 6やWR 7に書き込まれた同期キャラクタやフラグと一致するキャラクタが受信されるのを待ちます。一致すると，RR 0のビット4が'1'になるとともにE/S割り込みを発生します。非同期モード以外にプログラムしたときやアボートを受信したとき，レシーバがディセーブルされたときにはSCCは自動的にハントモードになります。

4 ビット3（受信CRCチェックイネーブル）

このビットは，受信キャラクタをCRC計算用のデータとして扱うか否かを制御するものです。'1'にすると，受信されたキャラクタがCRCの計算に含まれるようになります。非同期モードでは，このビットの設定は無視されます。

5 ビット2（アドレスサーチモード）

SDLCモードのときだけ有効なモードです。'1'にすると，SCCはSDLCのメッセージ中のアドレスフィールド値とWR 6に設定した値を比較し，一致しないメッセージを無視します。このとき，WR 3のビット1（SYNCキャラクタ・ロード禁止）を'1'にすると，SCCはアドレスフィールドの上位4ビットだけを比較するようになります。

6 ビット1（SYNCキャラクタ・ロード禁止）

SDLCモード以外の同期モードでは，SCCは，このビットを'1'にするとWR 6に設定されている値と受信データを比較し，一致したときにはそのデータを破棄します。破棄されたデー

タはCRC計算には含まれません。

Monosyncモードで同期キャラクタ長を6ビットとしても，SCCはあくまでも8ビット単位での比較しか行いませんので，注意が必要です。また，Bisyncモードで12ビットの同期キャラクタを指定すると，このビットの設定は無視されます。

SDLCモードでアドレスサーチモードを選択しているときに，このビットを'1'にしていると，アドレスフィールドとWR6の設定値の比較は上位4ビットだけで行われるようになり，上位4ビットが一致している局へのメッセージがすべて受信されます。

7 ビット0(受信動作イネーブル)

'1'にすると受信動作が許可，'0'で禁止になります。チャンネルリセットやハードウェアリセットが起こった場合，SCCはこのビットを自動的に'0'にします。

❶・❼5 WR4

ビット配置は図17のようになっています。このレジスタは，トランスミッタやレシーバの各種パラメータの設定を行うものです。

1 ビット7,6(クロック/データ転送速度比)

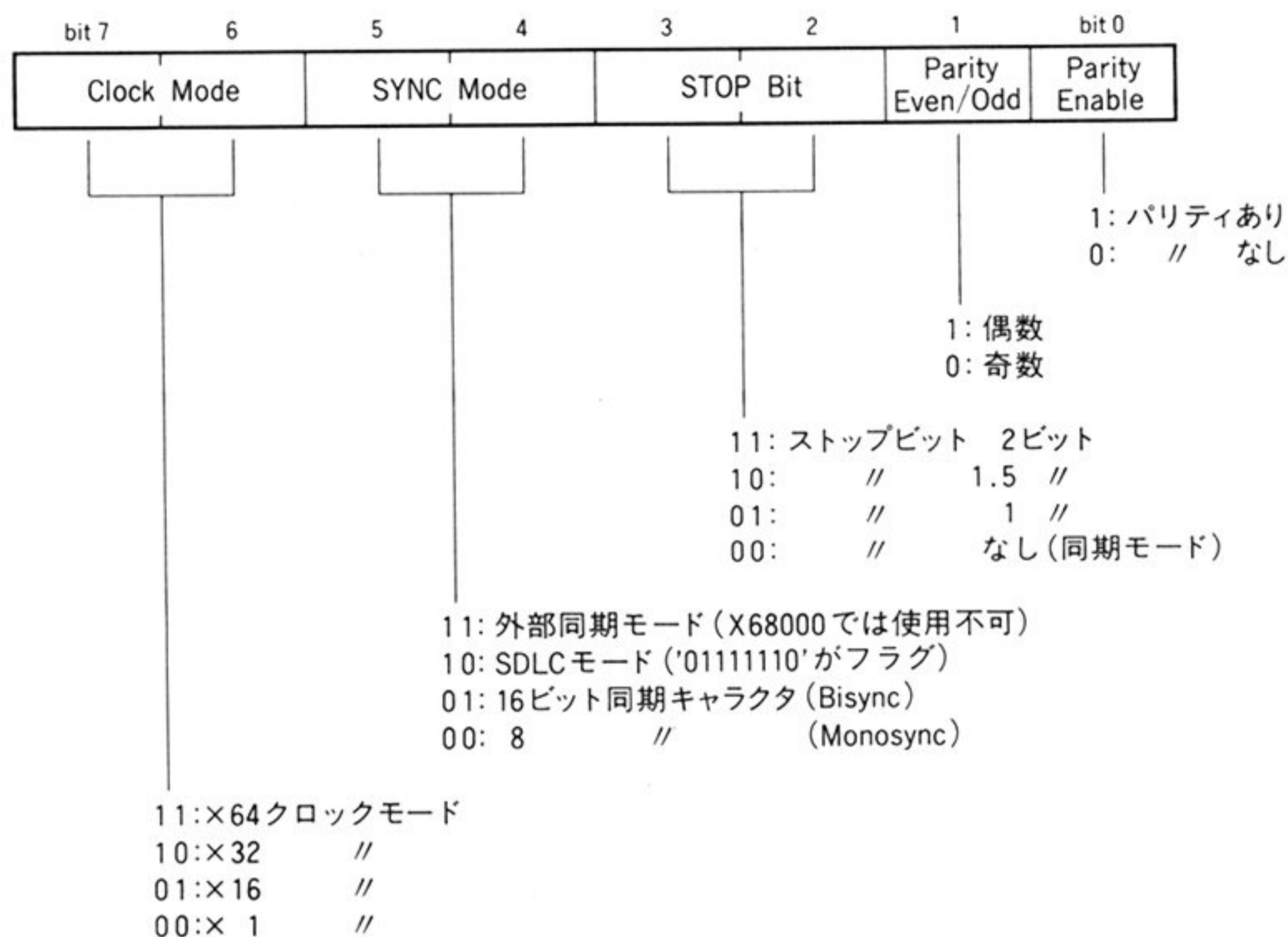
クロックとデータ転送速度の比率を決めます。たとえば，×16モードを選ぶと，データ転送速度は与えられたクロックの1/16になります。非同期モードでは，×1以外のモードを使用するようにします。

2 ビット5,4(同期モード)

受信データと同期をとる方法を指定します。ビット3，2が'00'，すなわち，同期モードが選択されていないときにはこれらのビットの設定は無効です。

11(外部同期モード)

SCCのSYNC端子の入力で同期をとるモードです。X68000ではSYNC端子は使用されていないので，このモードは使用できません。

●図……17 WR 4

10(SDLC モード)

SDLC モードでの動作になります。このとき,WR 7 にフラグデータ('01111110'),WR 6 にレシーバのアドレスを設定し,WR 5 によって CRC-CCITT を選択しなければなりません。

01(Bisync モード)

同期キャラクタは WR 6 と WR 7 を連結して設定します。同期キャラクタを 12 ビットと 16 ビットのいずれにするかは WR 10 のビット 0 で指定します。

00(Monosync モード)

同期キャラクタは WR 7 に設定します。SCC は同期キャラクタと同一のキャラクタを見つけて同期をとります。同期キャラクタ長は,WR 10 のビット 0 によって,6 ビット長と 8 ビット長のいずれかにするかを選択できます。

3 ビット3,2(ストップビット長)

非同期モードのときのストップビット長を指定します。同期モードを使用するときには,これらのビットは'00'に設定します。

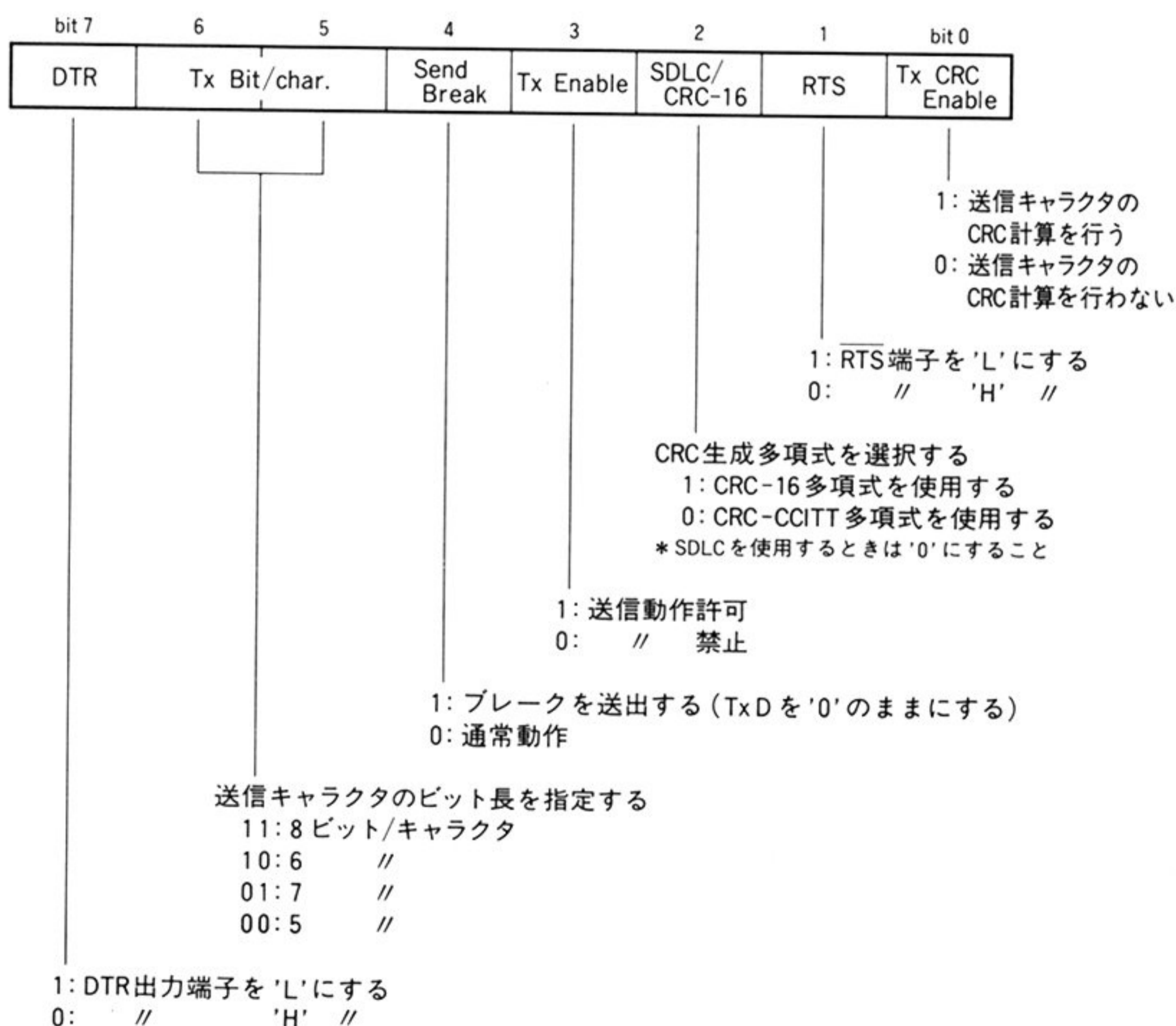
4 ビット1,0(パリティ選択)

非同期モードでのパリティビットの選択を行います。ビット0でパリティの有無を，ビット1でパリティを偶数パリティとするか，奇数パリティにするかを選択します。

キャラクタ長として8ビット未満を選択した場合，受信バッファにはパリティビットも取り込まれますので，注意が必要です。

❶·❼6 WR5

WR5は送信パラメータの設定と送信制御を行います。ビット配置は図18のようになっています。

●図……18 WR5

1 ビット7(DTR制御)

SCCのDTR信号の状態を操作します。このビットを'1'にすると，SCCのDTR出力端子が'Low'レベル（レディ状態）に，'0'にすると'High'レベルになります。

2 ビット6, 5(送信キャラクタビット長)

送信キャラクタのビット長を指定します。データは下位ビットから順に送出されていきます。

3 ビット4(ブレーク送出)

このビットを'1'にすると，TxDが'0'になります。この機能は，トランスミッタのイネーブル/ディセーブルに関係なく動作します。

Monosyncでループモードが選択された場合，レシーバで同期が確立すると，このビットは'0'になり，トランスミッタは同期キャラクタやデータの送信を開始します。SCCがチャンネルリセットやハードウェアリセットされた場合は自動的に'0'になります。

4 ビット3(送信イネーブル)

このビットを'0'にすると送信動作が行われなくなり，TxD端子は'1'になります。CRCキャラクタの送信中にこのビットが'0'になると，CRCのかわりに同期キャラクタやフラグが送信されます。

このビットは，SCCのチャンネルリセットやハードウェアリセットで'0'になります。

5 ビット2(CRC生成多項式選択)

送受信で使用するCRCの演算方法を選択します。'1'のときにはCRC-16多項式，'0'のときにはCRC-CCITT多項式が使用されます。SDLCモードではCRC-CCITT多項式を選択します。

CRCジェネレータとチェッカは，WR 10のビット7によって，全ビットを'1'と'0'のいずれかにプリセットすることができます。

6 ビット1（RTS制御）

SCCのRTS信号の状態を操作します。このビットを'1'にすると，SCCのRTS出力端子が'Low'レベル（レディ状態）に，'0'にすると'High'レベルになります。

7 ビット0（送信CRCイネーブル）

送信キャラクタのCRC計算をするか否かを指定します。'1'にすると送信キャラクタのCRC演算が行われ，送信アンダーランが発生するとCRCデータが送出されます。

❶・❼7 WR6/WR7

ビット配置を図19に示します。Monosync，Bisyncモードでは WR6，WR7に同期キャラクタを設定します。Bisyncモードでは，WR6に下位バイト，WR7に上位バイトを設定します。

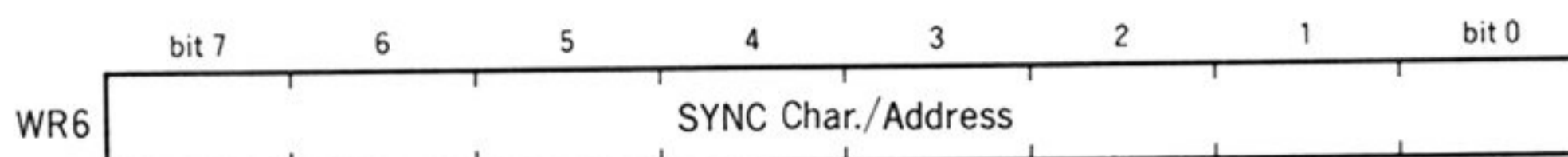
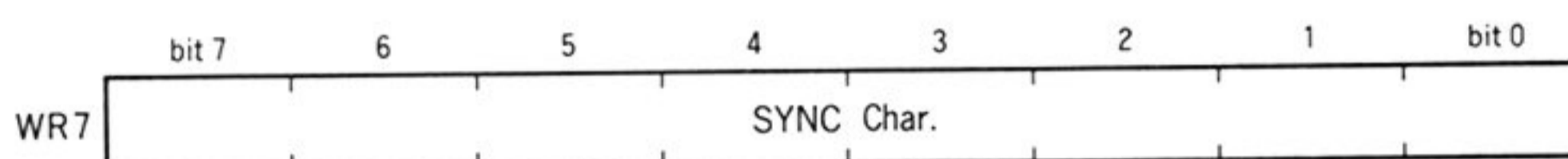
SDLCモードでは，WR6には自局のアドレス，WR7にはフラグキャラクタ('01111110')を設定します。

❶・❼8 WR9

WR9は割り込み制御などを行います。ビット配置は334ページの図20のようになっています。WR9は内部的には1つしかなく，いずれのチャンネルからアクセスされても同じものがリード/ライトされます。

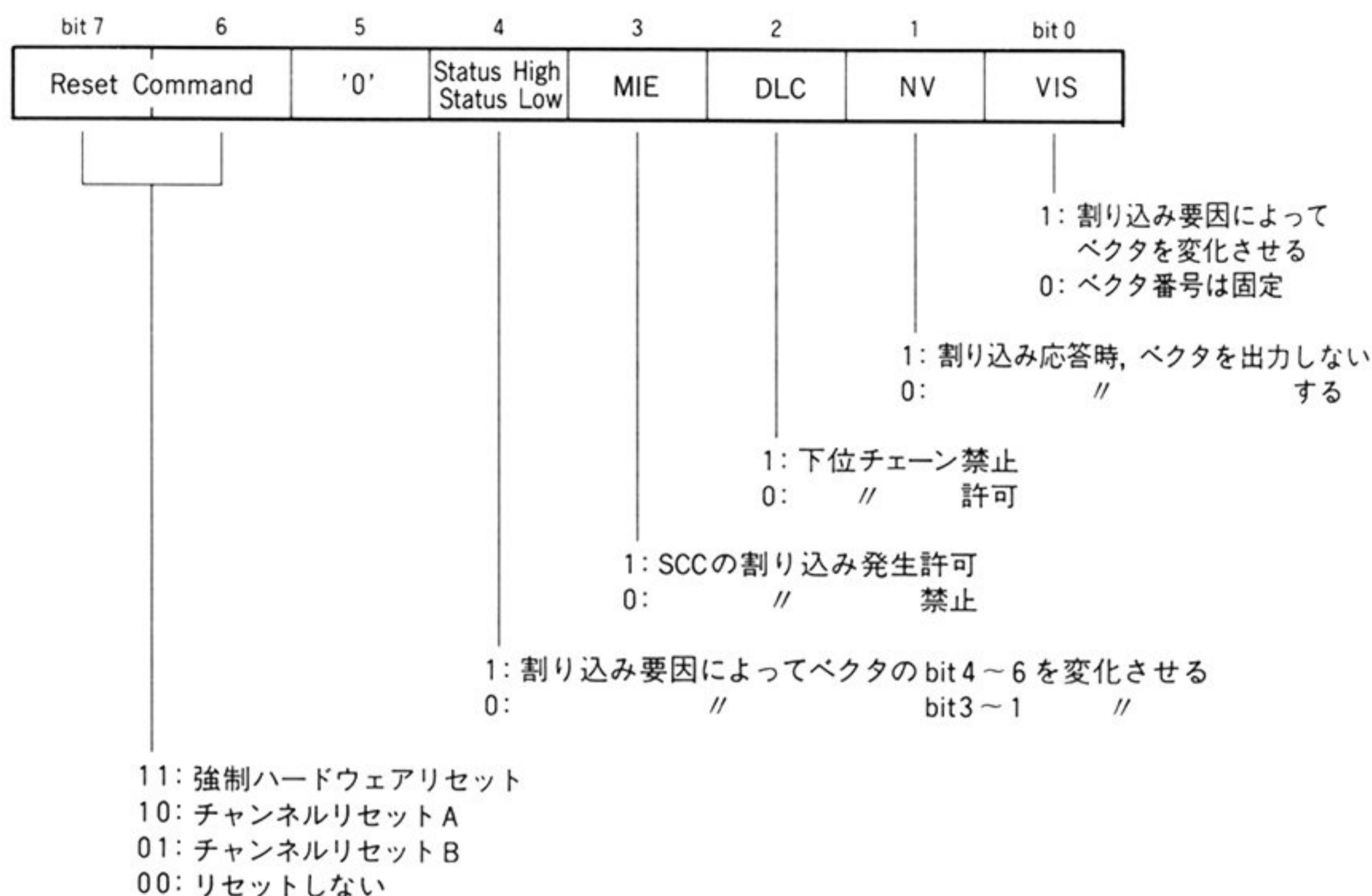
1 ビット7，6（リセットコマンド）

SCCの各チャンネルをリセットします。ビット7がチャンネルA，ビット6がチャンネルBに対応し，それぞれ'1'がセットされると，該当するチャンネルがリセットされます。'11'を設定したときは，WR0のビット0，1，WR9のビット2，3，4などが変化しないほかはハードウェアリセットと同様の働きをします。

●図……19　WR 6/WR 7

モード	WR6の値							
	bit 7	6	5	4	3	2	1	bit 0
Monosync 8bits	SYNC$_7$	SYNC$_6$	SYNC$_5$	SYNC$_4$	SYNC$_3$	SYNC$_2$	SYNC$_1$	SYNC$_0$
Monosync 6bits	SYNC$_1$	SYNC$_0$	SYNC$_5$	SYNC$_4$	SYNC$_3$	SYNC$_2$	SYNC$_1$	SYNC$_0$
Bisync 16bits	SYNC$_7$	SYNC$_6$	SYNC$_5$	SYNC$_4$	SYNC$_3$	SYNC$_2$	SYNC$_1$	SYNC$_0$
Bisync 12bits	SYNC$_3$	SYNC$_2$	SYNC$_1$	SYNC$_0$	'1'	'1'	'1'	'1'
SDLC	ADR$_7$	ADR$_6$	ADR$_5$	ADR$_4$	ADR$_3$	ADR$_2$	ADR$_1$	ADR$_0$
SDLC(Address0)	ADR$_7$	ADR$_6$	ADR$_5$	ADR$_4$				

モード	WR7の値							
	bit 7	6	5	4	3	2	1	bit 0
Monosync 8bits	SYNC$_7$	SYNC$_6$	SYNC$_5$	SYNC$_4$	SYNC$_3$	SYNC$_2$	SYNC$_1$	SYNC$_0$
Monosync 6bits	SYNC$_5$	SYNC$_4$	SYNC$_3$	SYNC$_2$	SYNC$_1$	SYNC$_0$		
Bisync 16bits	SYNC$_{15}$	SYNC$_{14}$	SYNC$_{13}$	SYNC$_{12}$	SYNC$_{11}$	SYNC$_{10}$	SYNC$_9$	SYNC$_8$
Bisync 12bits	SYNC$_{11}$	SYNC$_{10}$	SYNC$_9$	SYNC$_8$	SYNC$_7$	SYNC$_6$	SYNC$_5$	SYNC$_4$
SDLC	'0'	'1'	'1'	'1'	'1'	'1'	'1'	'0'

2 ビット4(ベクタ変更モード選択)

SCCは，割り込み要因によって発生するベクタ番号を変化させる機能があります。このとき，割り込み要因によって，ベクタのビット4～6を変化させるか，ビット3～1を変化させるかを選択するのがこのビットです。割り込み要因とベクタ番号の関係はRR 2のところを参照してください。

●図……20　WR 9

3 ビット3（割り込み発生許可/禁止）

'0'にすると，SCCからCPUへの割り込みの発生が禁止され，割り込み要求が発生しなくなります。このビットはハードウェアリセットで'0'になります。

4 ビット2（下位チェーン禁止）

SCCなど，Z 8000のファミリーLSIをデイジーチェーン接続し，1つの割り込み要求信号を複数のLSIで共有するような使い方をしたときに有効なものです。X 68000ではSCCを単独で使用していますので，このビットの操作は意味を持ちません。リセット後，このビットは'0' になります。

5 ビット1（ベクタなし）

このビットを'1'にすると，SCCは割り込みベクタの出力を行わなくなります。SCCを割り込みコントローラと接続し，割り込みベクタを割り込みコントローラに出力させるような場合には，SCCをこのモードにして，割り込みコントローラが出力するベクタとSCCが出力する

ベクタが衝突しないようにします。

6 ビット0(ベクタインクルードステータス)

割り込み要因によってベクタを変化させるか否かを選択します。'1'にすると割り込み要因によってベクタ番号が変化するようになり，'0'にすると割り込み要因によらず，つねにWR 2に書き込んだベクタ番号が出力されるようになります。

❶·❼ 9 WR 10

WR 10は送受信動作の制御用レジスタです。WR 10のビット配置を336ページの図21に示します。

1 ビット7(CRCプリセット)

CRCチェッカ/ジェネレータの初期値を，すべて'1'にするか'0'にするかを指定します。

2 ビット6, 5(データの符号化)

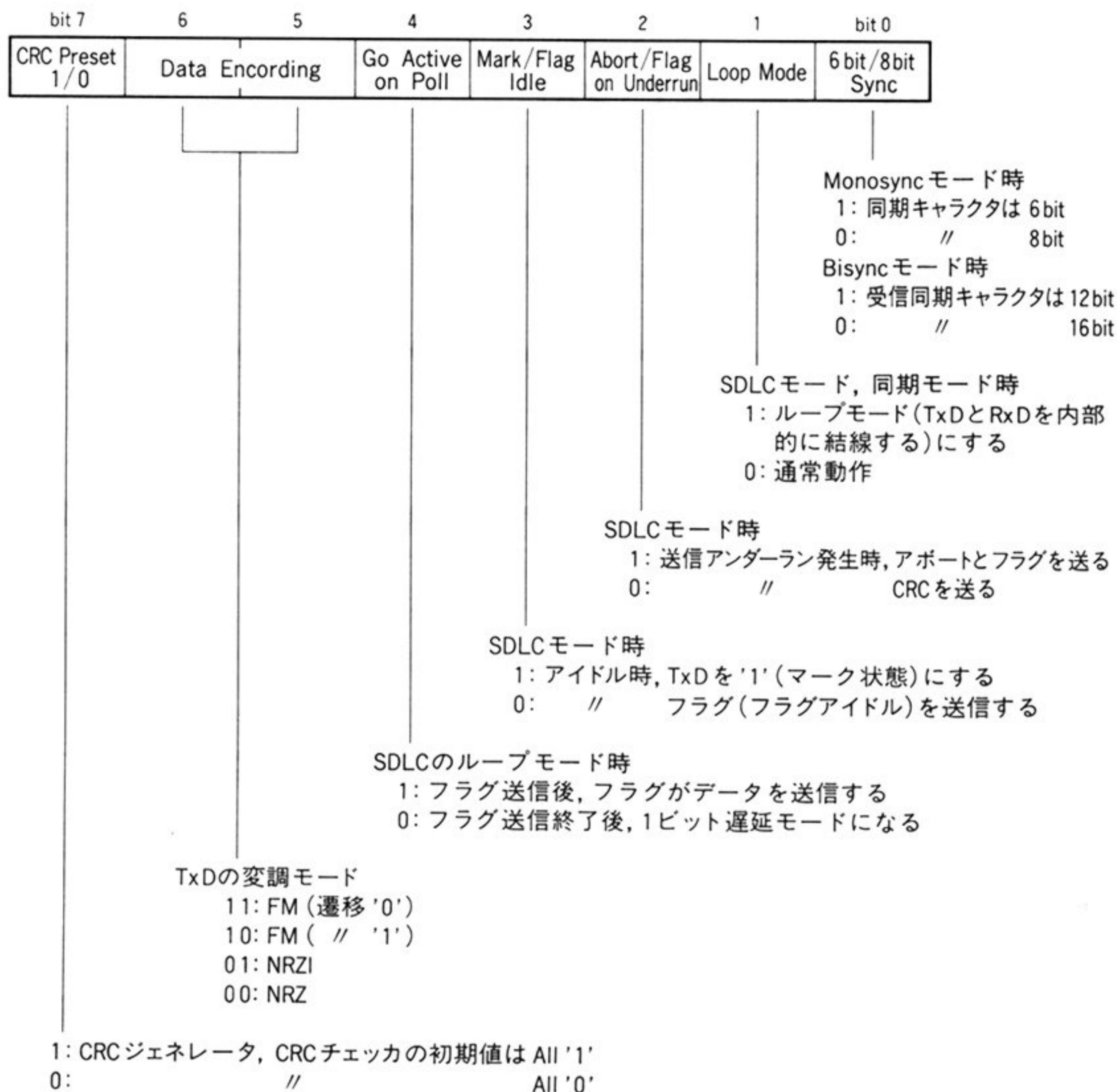
SCCが入出力するデータの符号化の方法を選択します。もっとも一般的に使用されているデータの'1'，'0'が，そのまま出力の'High'，'Low'に対応するのがNRZと呼ばれる符号化です。

その他のNRZIやFMモードでは，クロックとデータを分離するためにSCC内部のDPLLを使用することもできます。DPLLを使用するときには，WR 14のビット7，6，5でも，どの符号化を行うかを指定するようにします。

3 ビット4(ポーリングでアクティブ)

おもにSDLCループモードで動作するときに使用されるビットです。このビットが'1'になっているときにEOPが受信されると，オンループとなり，トランスミッタがイネーブルになります。フラグが送信された時点でこのビットが'1'になっていると，SCCは次のフラグやデータの送信を行い，'0'であればフラグ送信を完了後，通常の1ビット遅延モード(RxDから入力されたデータを1ビット分の遅延後，TxDから再送信する）になります。

SDLC以外の同期ループ伝送モードなら，トランスミッタが受信同期キャラクタに応答して

●図……21　WR 10

アクティブになる前に，このビットを'1'にしなくてはなりません。

4 ビット3(マーク/フラグ・アイドル)

SDLCモードのときだけ有効なモードで，アイドル時のTxDの状態の制御を行うものです。このビットを'0'にすると，SCCはアイドル時にフラグを送信します。'1'にするとアイドル時はフレーム終了フラグを送出した後，TxDは'1'のままになります。

5 ビット2(アンダーランでアボート/フラグ)

このビットもSDLCモードのときだけ有効です。SCCが送信アンダーランのときの動作を

選択するものです。

'0'にすると，送信アンダーランが発生したときにCRCデータを送信し，'1'にするとアボートとフラグを送信します。同時に，RR 0のビット6（送信アンダーラン/EOM）が'1'となり，E/S（外部/ステータス）割り込みが発生します。さらにCRC送出が終わると，TxDは'1'に固定されるとともに送信バッファ空の割り込みが発生します。

通常，SDLCモードの場合には，データの先頭バイトを書き込んだ後に'1'にし，最終バイトを書き込んだ後に'0'にするようにします。

SDLCループモードでは，このビットは無効になります。

6 ビット1（ループモード）

SCCをループモードにします。トランスミッタとレシーバをイネーブルにするのは，このビットを設定した後に行います。

SDLCモードでは，WR 10のビット4が'1'にセットされた後，EOPが受信されると，SCCはオンループになりますが，その後，このビットが'0'に戻されると，次のEOPでSCCはループを離れます。

SDLC以外の同期モードでは，レシーバとトランスミッタを同期させるために使用します。レシーバは同期キャラクタを受け取ると，そのキャラクタ境界でトランスミッタをイネーブルにします（TxDをブレーク状態にしていても解除されます）。

このビットは非同期モードでは無視されます。

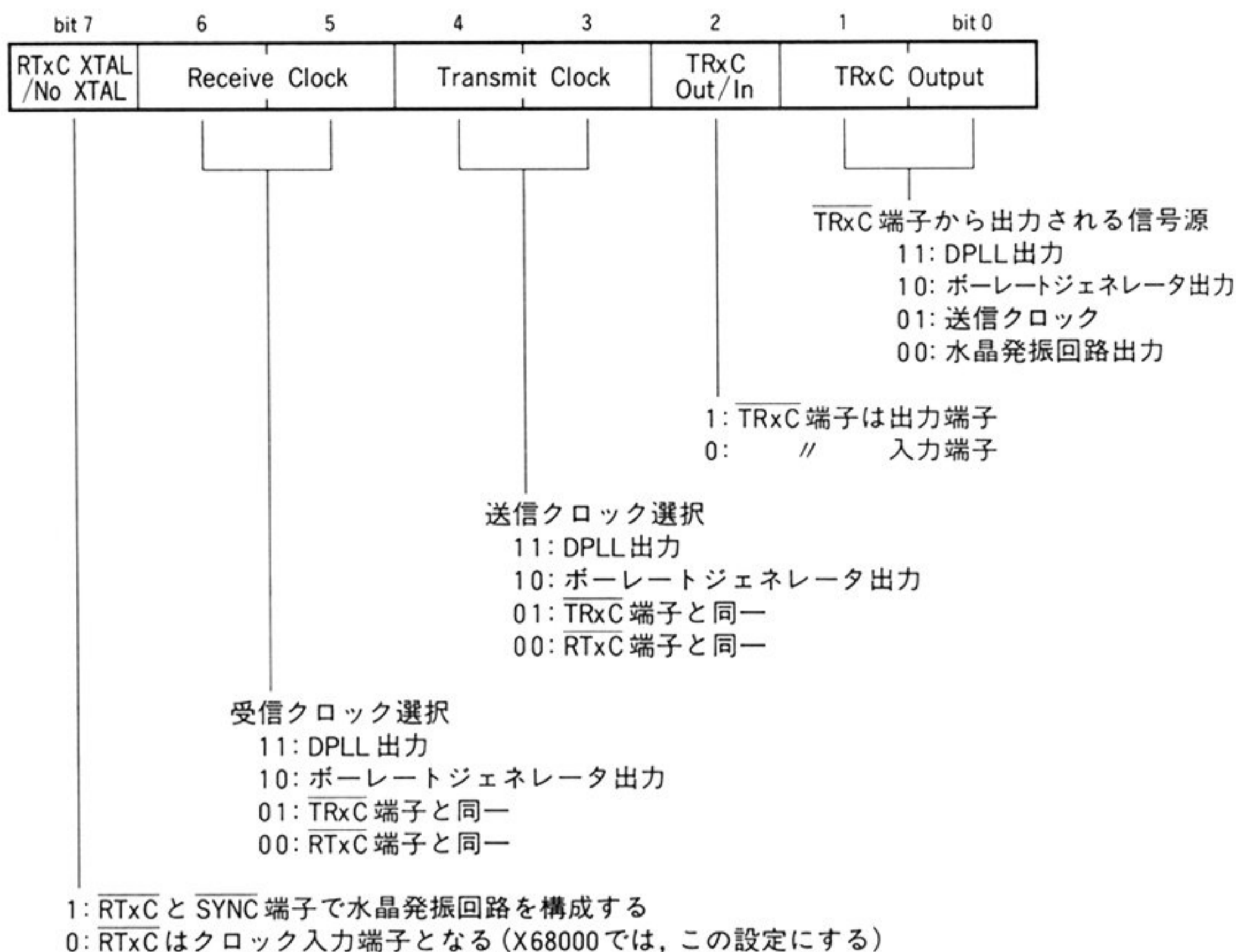
7 ビット0（同期キャラクタ長）

MonosyncやBisyncモードのときに，同期キャラクタ長を通常の8ビット（Monosync）や16ビット（Bisync）ではなく，6ビットや12ビットとするために使用されます。

このビットへの設定は，SDLCモードや非同期モードでは無視されます。

❶・❼ 10 WR11

WR 11は，送受信タイミング用クロックやSYNC端子の機能の選択などを行います。ビット配置は338ページの図22のようになっています。

●図……22　WR11

1 ビット7(RTxC水晶あり/なし)

SCCは，RTxC端子とSYNC端子の間に水晶振動子を接続すると，発振回路を構成し，自分で発振動作を行うことができるようになっています。このビットが'1'になっていると，SCCはSYNC端子との間に接続されている水晶振動子を使って発振動作を行うようになり，'0'にすると，RTxCは外部からのクロック入力端子となります。

X 68000では，チャンネルA，チャンネルBとも'0'で使用するようにします。

2 ビット6, 5(受信クロック源選択)

受信動作のクロック源の選択を行います。通常の非同期モードでは'10'，すなわち，ボーレートジェネレータの出力を使用します。ハードウェアリセット後は，受信クロックはRTxCから供給されるモードになっています。

3 ビット4,3(送信クロック選択)

送信動作のクロック源の選択を行います。通常の非同期モードでは'10',すなわち,ボーレートジェネレータの出力を使用します。ハードウェアリセット後,送信クロックはTRxCから供給されるモードになっています。

4 ビット2(TRxC出力/入力)

SCCのTRxC端子をクロック入力端子として使うか,クロック出力端子として使うのかを選択します。レシーバやトランスミッタのクロック源としてTRxC端子を選択している場合には,TRxC端子はこのビットの設定に関係なく,強制的に入力端子となります。

X68000では,チャンネルAのTRxC端子は入出力のいずれでも使用できるようになっています。入力端子として使うときにはチャンネルBのDTR端子を'1'('Low'レベル)にすると,RS-232CコネクタのST2(送信タイミング入力)端子がTRxC端子への入力となります。TRxC端子を出力として使うときにはチャンネルBのDTR端子を'0'('High'レベル)にしておかないと,ST2からの入力とSCCの出力が衝突してしまいますので注意してください。

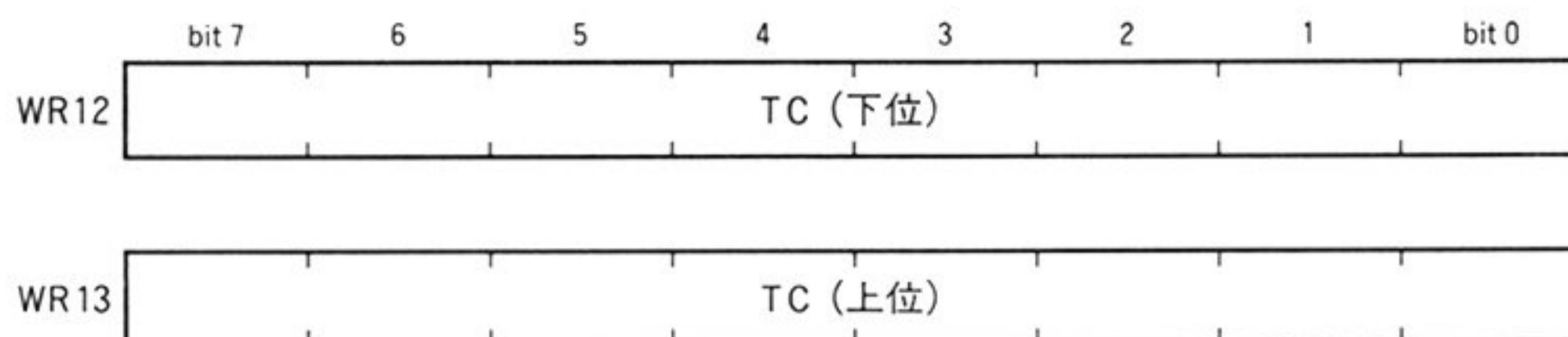
5 ビット1,0(TRxC出力源)

TRxC端子が出力端子として動作しているとき,この端子から出力されるクロック源を選択します。DPLL出力を選択した場合,TRxC端子から出力されるのは受信用のDPLLが生成しているクロックです。

❶・❼11 WR12/WR13

ボーレートジェネレータの出力周波数制御を行うレジスタです。ビット配置は340ページの図23のようになっており,WR12が下位8ビット,WR13が上位8ビットの16ビットレジスタとして動作します。

これらのレジスタへの書き込みの際は,ボーレートジェネレータの動作をいったん停止させてから行うようにします。

●図……23　WR 12, WR 13

$$\text{ボーレートジェネレータ出力周波数} = \frac{\text{ボーレートジェネレータへの入力クロック周波数}}{2 \times (TC + 2)}$$

❶・❼ 12　WR 14

WR 14 はボーレートジェネレータや DPLL の制御などに使用されます。ビット配置は図 24 のようになっています。

1 ビット7, 6, 5（DPLLコマンド）

DPLL の動作モードの選択などを行います。

111（NRZI モード選択）

DPLL を NRZI 符号のデコード用として動作させます。リセット後，DPLL はこのモードになります。

110（FM モード）

DPLL を FM 符号やマンチェスター符号のデコード用として動作させ，入力された FM 符号信号に同期したクロックを生成します。

101（DPLL クロック源＝RTxC）

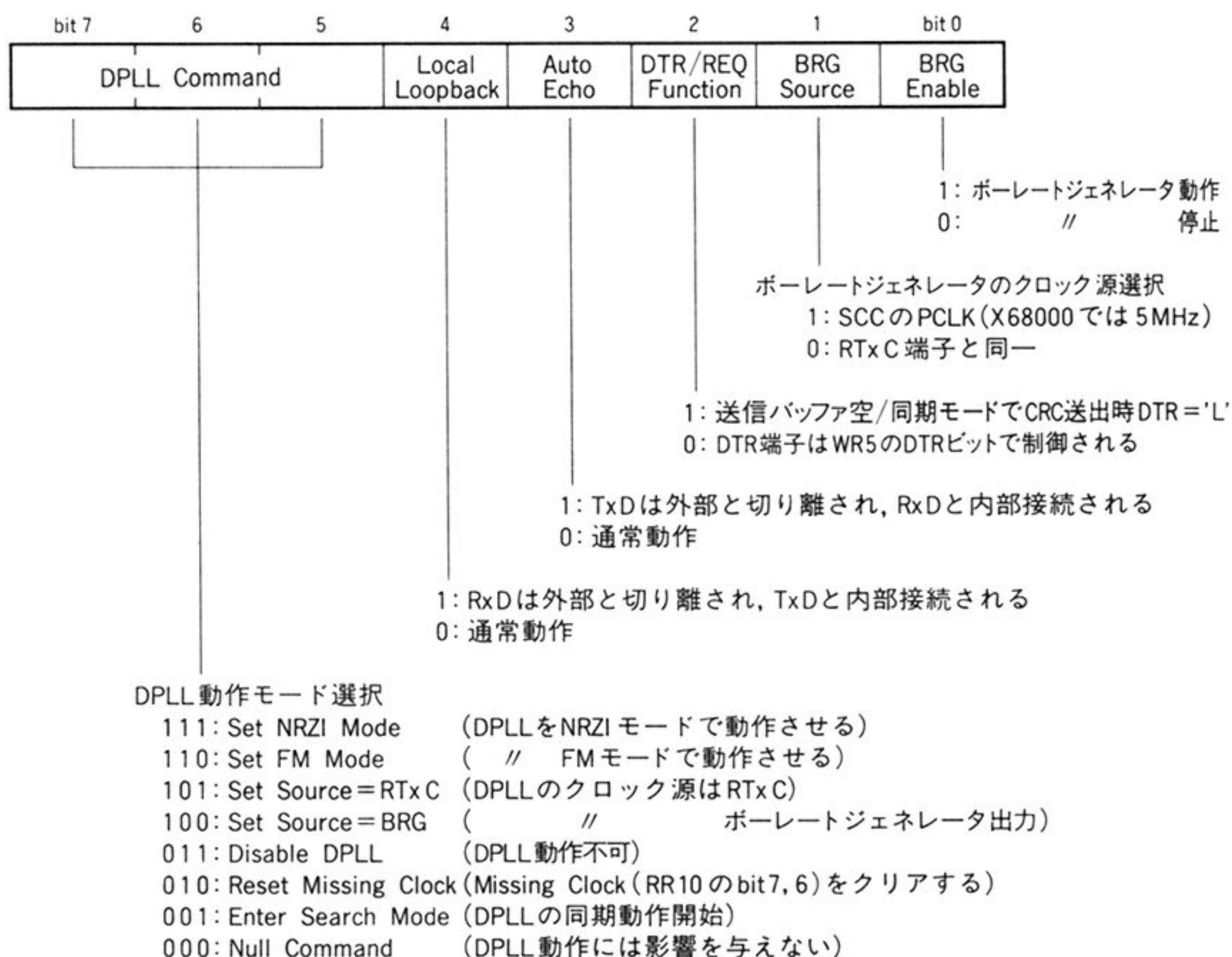
DPLL のクロック源として RTxC 端子の入力を使います。

100（DPLL クロック源＝BRG）

DPLL のクロック源としてボーレートジェネレータの出力を使用します。DPLL を NRZI モードで動作させる場合には，ボーレートジェネレータのクロックは伝送速度の 32 倍，FM モードで動かす場合には 16 倍のクロックが入力されるようにボーレートジェネレータをプログラムする必要があります。

011（DPLL ディセーブル）

DPLL の動作を停止させます。クロック欠如ビット（RR 10 のビット 7， 6）はクリアされ，

●図……24　WR14

サーチモードになります。

010（クロック欠如リセット）

クロック欠如ビットをクリアし，次のクロック欠如状態が検出されるようになります（クロック欠如ビットはFMモードでのみ使用されます)。

001（エンターサーチモード）

このコマンドを受け取ると，DPLLはサーチモードになり，入力データに同期をとるようになります。FMモード時，決められた期間内に入力信号のエッジが検出できないと，「1クロック欠如」となり，RR 10のビット7が'1'になります。さらに連続して2回試みても入力信号のエッジが検出できなければ，「2クロック欠如」となり，RR 10のビット6が'1'になるとともにDPLLはサーチモードになります。

000（ヌルコマンド）

DPLLの動作にはなんら影響を与えません。

2 ビット4（ローカルループバック）

'1'にすると，SCCはローカルループバックモードになり，トランスミッタの出力はそのままレシーバにも入力され，RxD端子は使用されなくなります。

リセット後，このビットは'0'になります。

3 ビット3（オートエコー）

'1'にすると，SCCはオートエコーモードになり，RxDへの入力はそのままTxDからも出力されるようになり，トランスミッタの出力は無視されます。

リセット後，このビットは'0'になります。

4 ビット2（DTR/REQ機能選択）

SCCのDTR/REQ端子を，ソフトウェアで操作可能であるDTR信号として使用するか，データ転送要求信号として使用するかを決めます。'1'のとき，この端子はDTR信号端子となり，WR 5のビット7で状態を設定することができます。このビットを'0'にすると，この端子は転送要求信号となり，送信バッファが空になったときや，同期モードでCRCデータの送出が行われた時点で，この端子が'Low'になります。

X 68000では，DTR信号としてRS-232 Cコネクタに出力していますので，通常，このビットは'0'で使用します。

5 ビット1（BRGクロック源）

ボーレートジェネレータのクロック信号源としてRTxC端子への入力を使用するか，SCCの基本クロック（PCLK端子から入力される）を使用するかを選択します。'1'のとき，PCLK入力が選択されます。

通常，非同期モードでは，外部からクロックは与えられませんので，'1'で使用するのが普通でしょう。X 68000ではPCLK端子に5 MHzのクロック信号が入力されています。

6 ビット0（BRG動作イネーブル）

BRGの動作の許可/禁止を制御します。'1'のとき，ボーレートジェネレータの動作がイネーブルになります。

WR 12, WR 13に設定を行う場合には，このビットを'0'にしてボーレートジェネレータの動作を停止させ，設定が終了してから，'1'に戻すようにします。

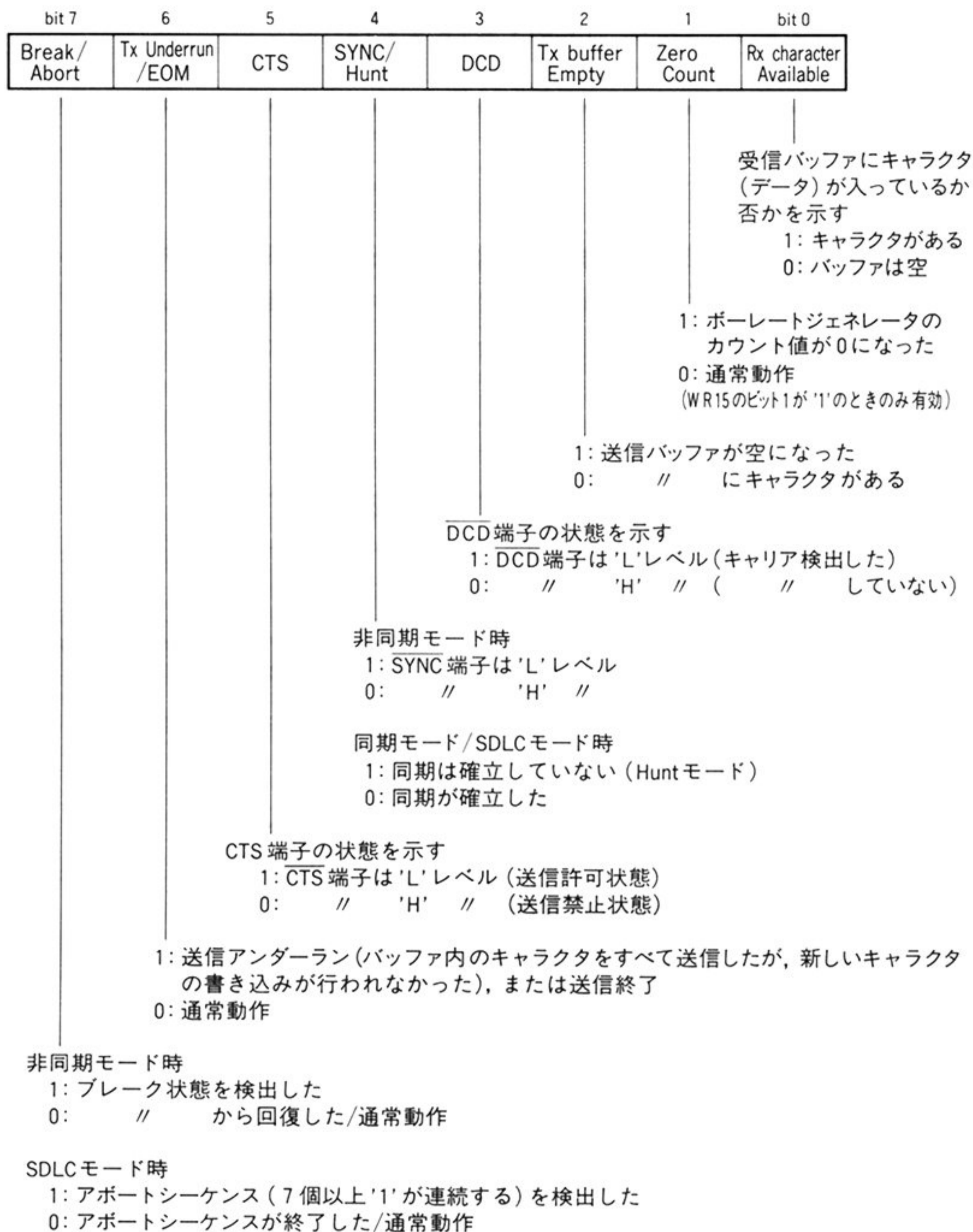
❶・❼13 WR15

WR 15のビット配置を図25に示します。WR 15は，E/S (外部/ステータス) 割り込み要因となりえるものそれぞれについて，割り込みを発生するか否かを選択するものです。それぞれ'1'になっていると割り込み発生が許可，'0'になっていると禁止となります。

●図……25　WR 15

❶・❼14 RR0

RR 0のビット配置を344ページの図26に示します。RR 0は，送受信バッファのステータスと6つのE/S割り込み要因ごとのステータスが示されています。

●図……26 RR 0

1 ビット7(ブレーク/アボート/EOP)

非同期モードでは,RxDにブレーク状態を検出すると,'1'になります。RxDが復帰すると,このビットが'0'になるとともにヌルデータ($00)が読み出されます。このデータは読み捨てる

必要があります。

SDLC モードでは，このビットは，アボートシーケンス（'1'が7個以上連続する）を検出した時点で'1'になり，アボートシーケンスが終了した時点で'0'になります。

このビットが'0'から'1'に変化した時点でE/S割り込みが発生します。

2 ビット6（送信アンダーラン/EOM）

リセットやトランスミッタディセーブル，アボート送出コマンドなどによって'1'になり，E/S割り込みが発生します。

このビットは，WR 0に「送信アンダーラン/EOM ラッチリセットコマンド」を書き込むことで'0'に復帰します。

3 ビット5（CTSラインステータス）

CTS (Clear To Send) 端子の状態を示します。WR 15のビット5でCTSの変化による割り込みがイネーブルになっている場合には，いずれかのE/S割り込み要因が発生したときのCTSの状態を保持し，CTSの状態に変化があれば，E/S割り込みが発生します。CTSによる割り込みが禁止になっていれば，このビットはCTS端子の状態がそのまま読み出されます。

4 ビット4（シンク/ハント）

非同期モードでは，SYNC端子の状態が示されます。X 68000では，SYNC端子は'High'レベルに固定されており，なんら有効なステータスにはなっていません。

SDLC モードでは，エンターハントコマンドが書き込まれたり，レシーバが動作不可になった場合に'1'となり，第1フレームの開始フラグが検出されると'0'になります。このとき，WR 15のビット4が'1'になっていれば，E/S割り込みが発生します。

5 ビット3（DCDラインステータス）

DCD端子の状態を示します。WR 15のビット3でDCDの変化による割り込みがイネーブルになっていれば，いずれかのE/S割り込み要因が発生した時点のDCDの状態を保持し，DCDの状態に変化があれば，E/S割り込みが発生します。DCDによる割り込みが禁止されていれば，このビットはDCD端子の状態がそのまま読み出されます。

6 ビット2(送信バッファ空)

送信バッファが空になると,'1'になります。このビットは,同期モードやSDLCモードではCRC送信中も'0'のままになっています。このビットはリセットによって'1'になります。

7 ビット1(ゼロカウント)

WR 15のビット1が'1'のとき,ボーレートジェネレータのカウント値が0になると,このビットが'1'になるとともにE/S割り込みを発生します。非同期モードなどでクロック源としてボーレートジェネレータを使用している場合には,この割り込みを使用しないのが普通でしょう。

8 ビット0(受信キャラクタ有効)

受信バッファに少なくとも1つのキャラクタが入っていると'1'になり,受信バッファが空になると'0'になります。リセットによって受信バッファは空になります。

1・7 15 RR 1

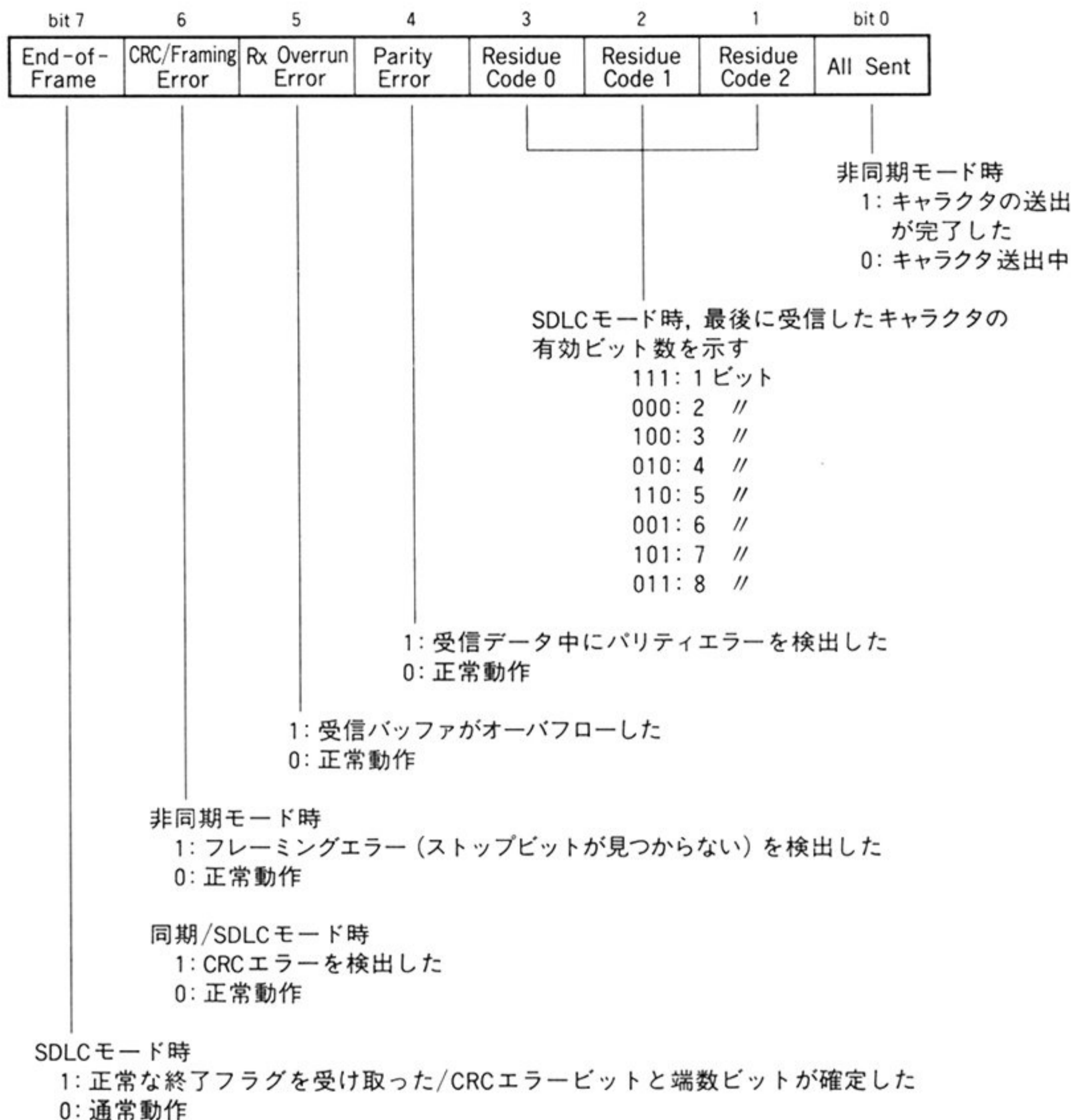
RR 1のビット配置を図27に示します。このレジスタの上位4ビットは,スペシャルRxコンディションのステータスビット,下位4ビットにはSDLCモード時の端数ビットなどが格納されます。

1 ビット7(エンドオブフレーム)

SDLCモード時のみ使用されます。正常な終了フラグを受け取ったときや,CRCエラービット,端数コードが確定したときに'1'になり,エラーリセットコマンドや後続の第1フレームが受信されたときに'0'に復帰します。

2 ビット6(CRC/フレーミングエラー)

非同期モードでフレーミングエラー(ストップビットがあるはずのところが'0'になっている)が発生した場合に,同期モードでは,このビットはCRCチェックの結果を示し,CRCエラー

●図……27　RR 1

が発生すると，'1'になります。

このビットは，エラーリセットコマンドや正常なキャラクタの受信によって'0'に復帰します。

3 ビット5（オーバーランエラー）

受信オーバーラン，すなわち，受信バッファがいっぱいになっているときに新しいキャラクタが受信された場合，オーバーランを起こしたキャラクタが引き取られた時点で'1'になります。この割り込みが発生したときは，エラーリセットコマンドを発行しないと，以後受信キャラクタが入ってくるたびにスペシャル Rx コンディション割り込みが発生してしまいます。

4 ビット4(パリティエラー)

パリティがイネーブルになっている場合，パリティチェックでエラーが検出されると，'1'になります。一度エラーが検出されると，エラーリセットコマンドでリセットするまで'1'のままになります。

WR 1のビット2によって，パリティエラーでスペシャルRxコンディション割り込みを発生させるようになっていると，パリティエラーを発生したキャラクタで割り込みを発生します。この割り込みが発生したときもエラーリセットコマンドを発行しないと，以後，受信キャラクタが入ってくるたびにスペシャルRxコンディション割り込みが発生してしまいます。

5 ビット3, 2, 1(端数コード)

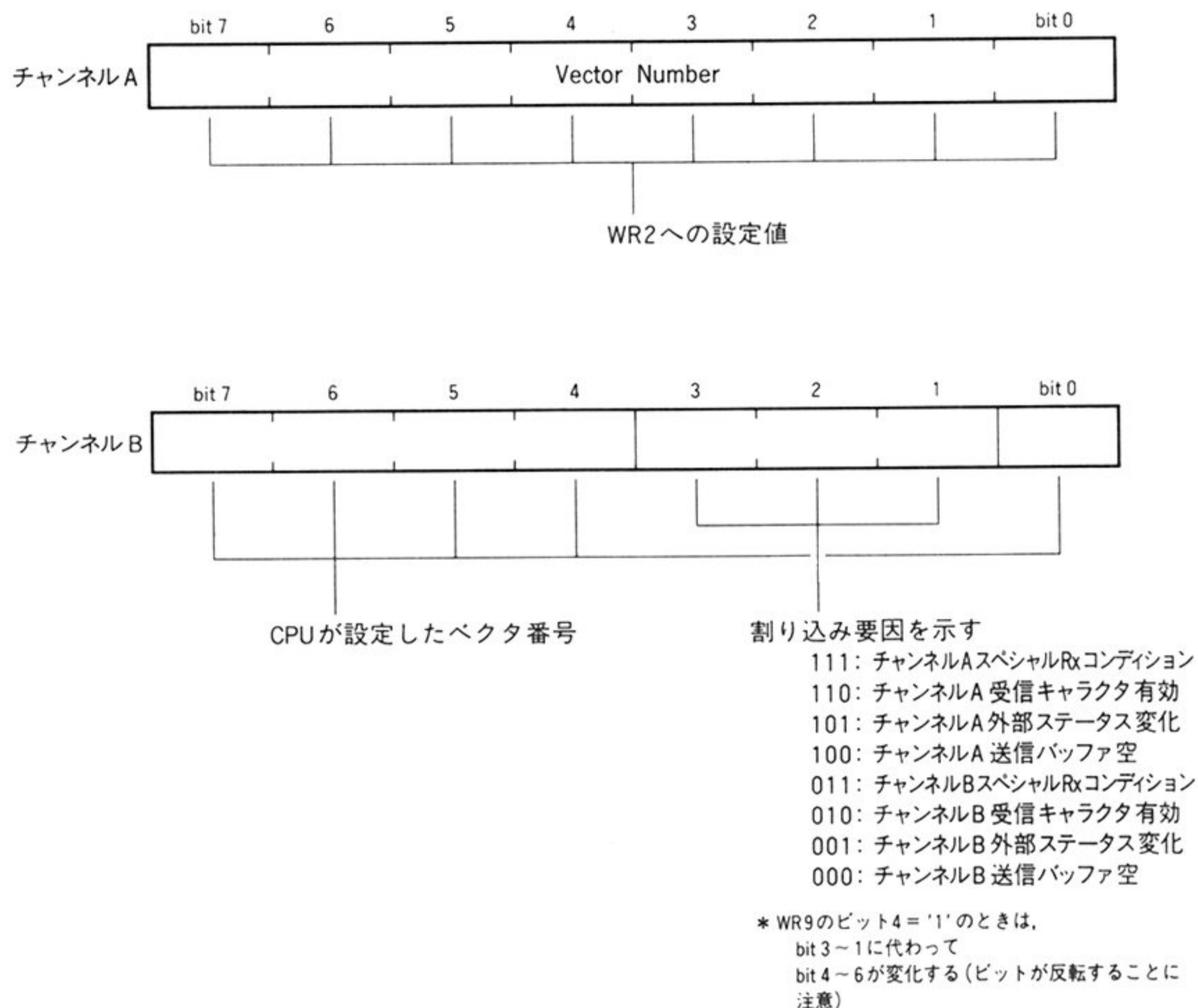
SDLCモードでは，データは任意のビット数のデータが伝送でき，SCCはこれを8ビットずつ受信バッファに取り込んでいきます。このため，最後のデータの有効ビットは1～8ビットのいずれかになります。この有効ビット数を示すのが端数ビットです。

6 ビット0(全キャラクタ送出)

非同期モードでは，このキャラクタがすべて送出されたときに'1'になります。送信バッファエンプティと似ていますが，送信バッファが空になった時点というのは，前回書き込んだキャラクタの送出が始まり，バッファに空きができたことを示すものであり，このビットは送出まですべて終了したことを示すものですので，間違えないようにしてください。

❶・❼16 RR2

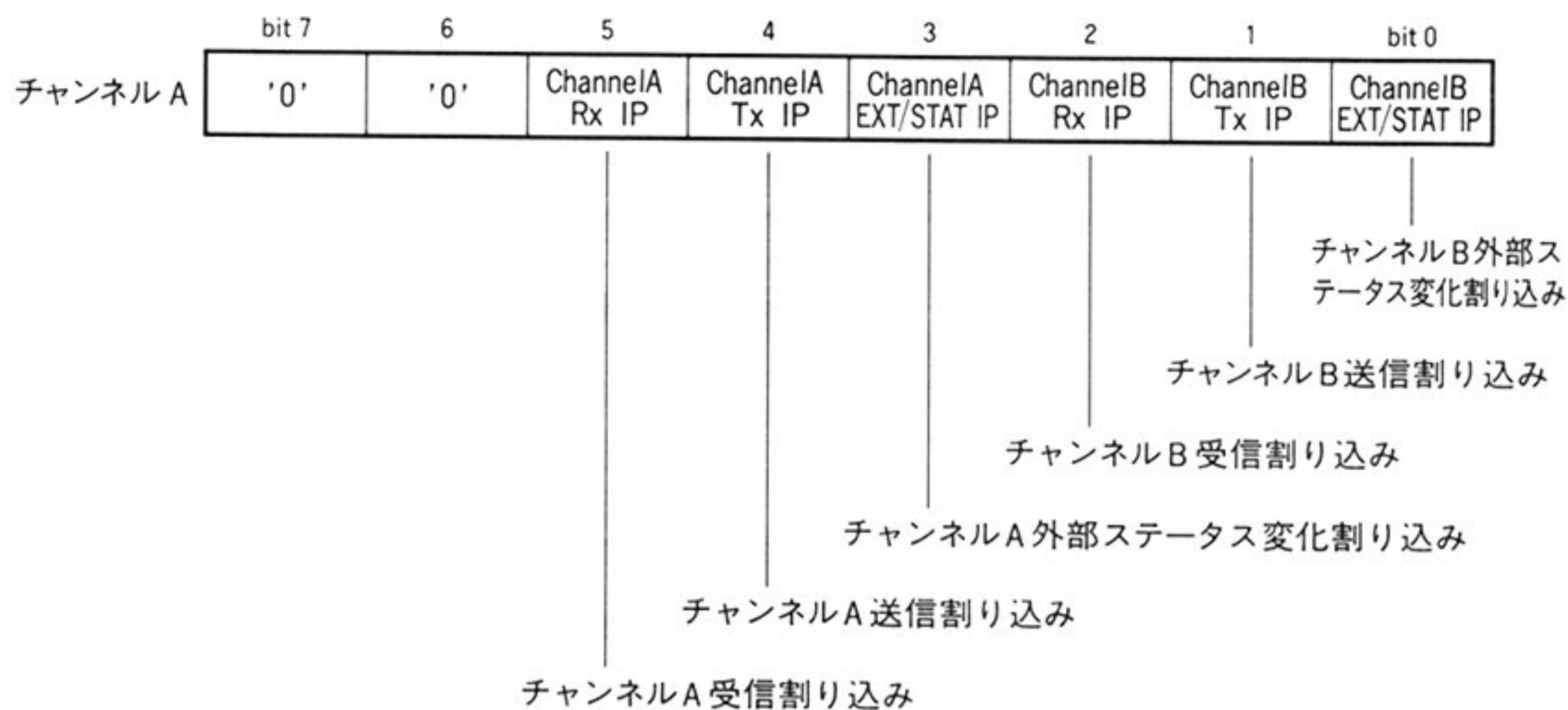
チャンネルA側ではWR 2に書き込んだベクタ番号そのものが，チャンネルB側には割り込み要因によって値が変化させられたベクタ番号がセットされます。割り込み要因によってビット4～6が変化するようにするモードと，ビット1～3が変化するようにするモードがあることはすでに述べたとおりですが，図28では，Human 68 Kで使用されているビット1～3が変化するモードでのベクタと割り込み要因の対応を示しています。

●図……28 RR 2

❶·❼17 RR 3

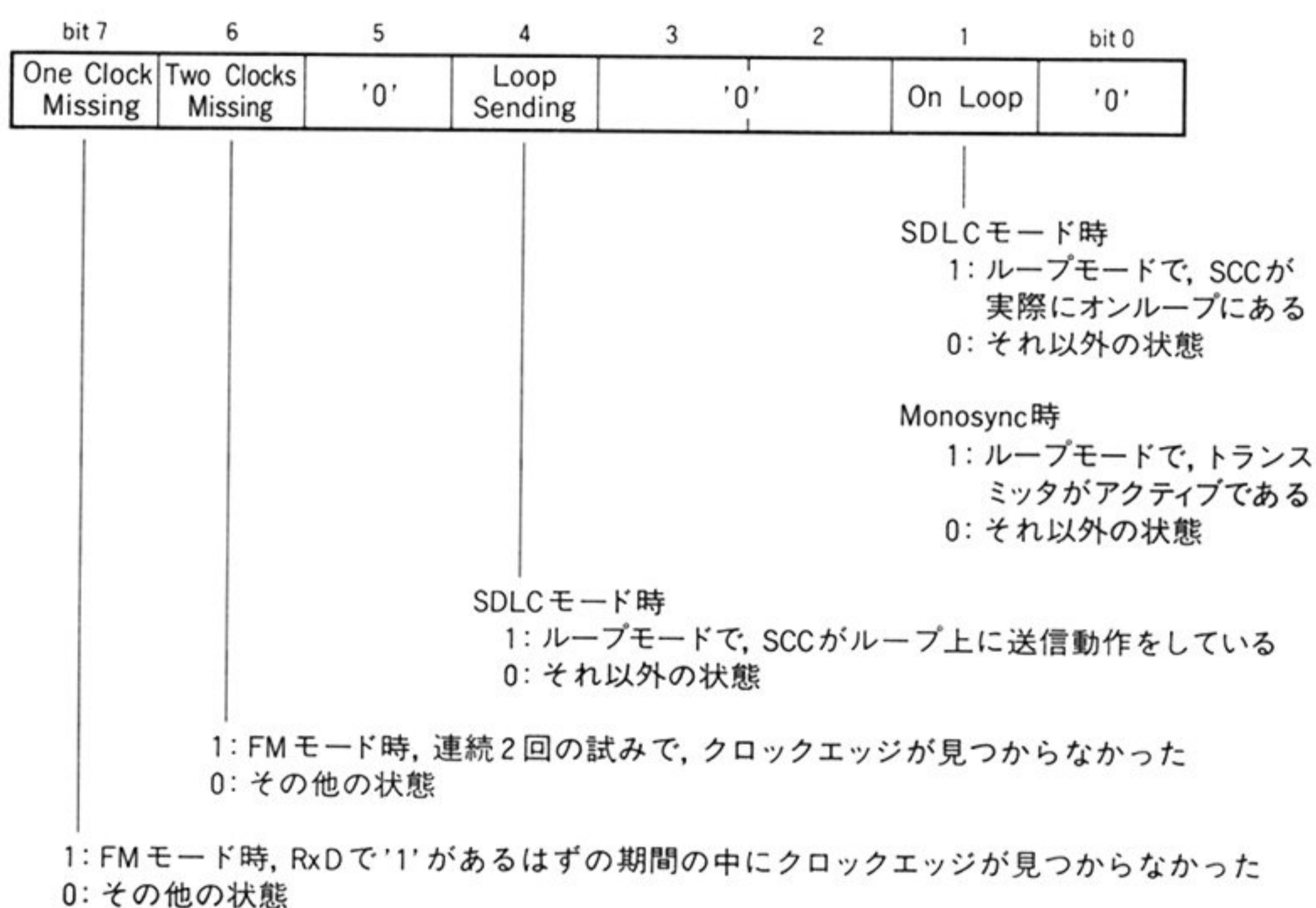
レジスタのビット配置を350ページの図29に示します。このレジスタはペンディング（保留）中になっている割り込み要因を示します。このレジスタは，チャンネルAのみが持っており，チャンネルBを読み出すと，$00が読み出されます。

❶·❼18 RR10

ビット配置を350ページの図30に示します。このレジスタには他のレジスタに入れられなかったステータスが集められています。

●図……29　RR 3(チャンネルAのみ有効)

＊いずれのビットも割り込みがペンディング(保留)中だと'1'になる

●図……30　RR 10

1: FMモード時，RxDで'1'があるはずの期間の中にクロックエッジが見つからなかった
0: その他の状態

1 ビット7, 6(クロック欠如)

FMモードで，DPLLが入力波形にエッジがあるはずの期間にエッジを検出できないと，ビ

ット7が'1'になり，連続2回の試みでもエッジが見つからないと，ビット6が'1'になります。

2 ビット4(ループ送信中)

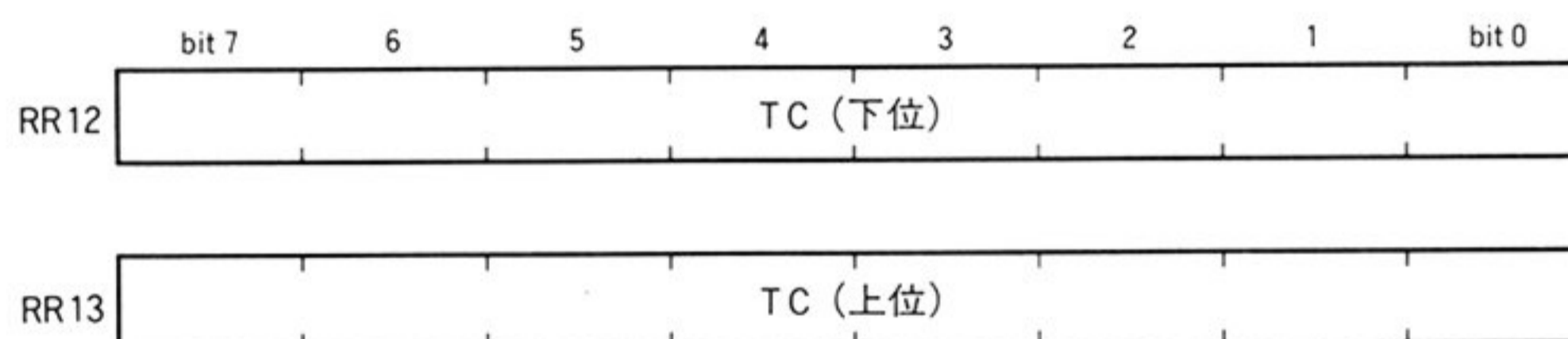
SDLCループモードで，トランスミッタがループの制御下にあり，SCCが送信動作をしている期間だけ'1'になります。

3 ビット1(オンループ)

SDLCループモードでは，SCCが実際にオンループにある期間，'1'になります。Monosyncでループモードに設定した場合には，トランスミッタがアクティブである間，'1'になります。

1・7 19 RR12/RR13

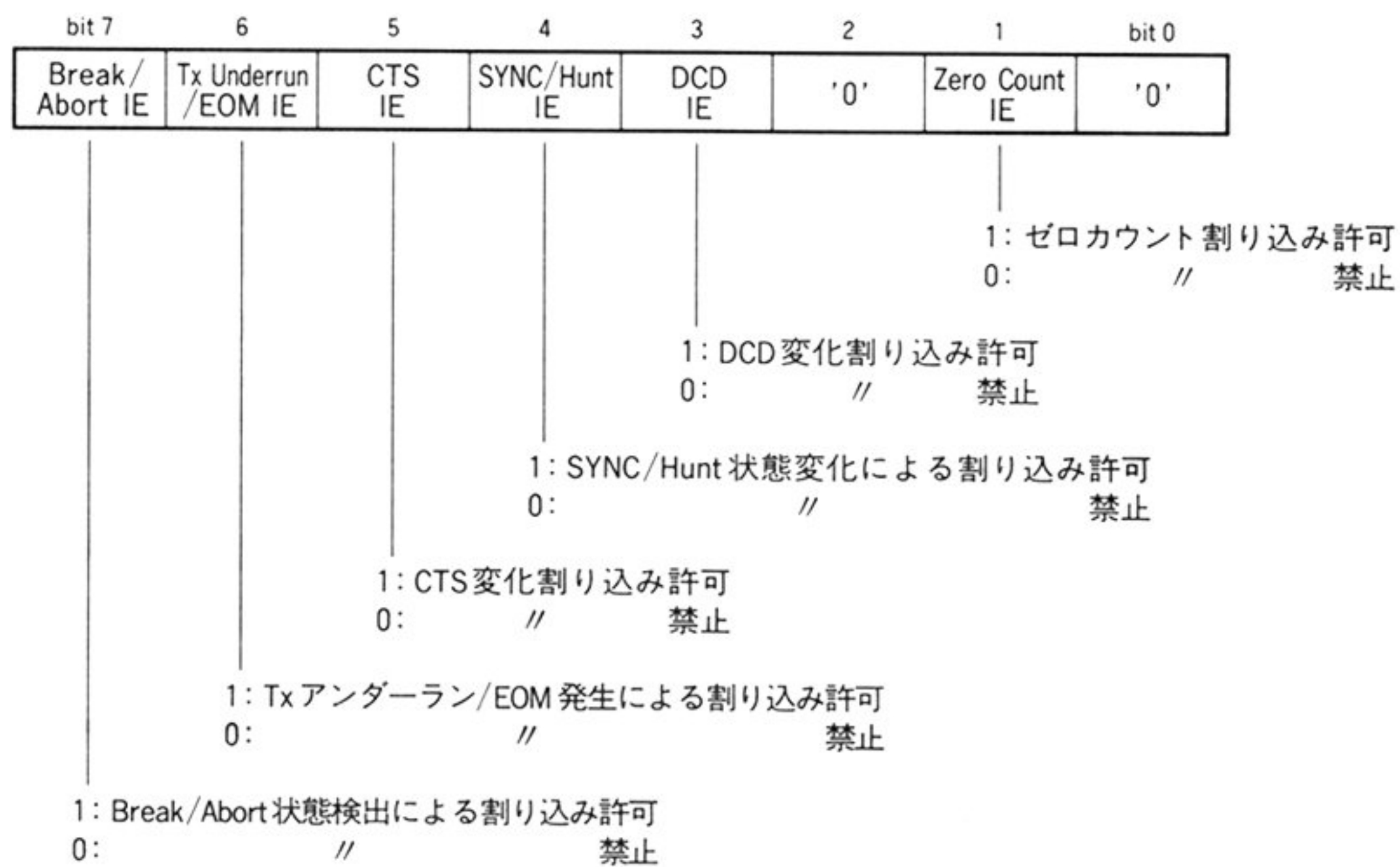
ビット配置を図31に示します。これらのレジスタは，ボーレートジェネレータ (WR 12/WR 13) に設定した値がそのまま読み出されます。

●図……31 RR 12, RR 13

※ WR12，WR13に書き込んだボーレートジェネレータへの設定値が読み出される

1・7 20 RR15

ビット配置を352ページの図32に示します。このレジスタは，WR 15に書き込んだ値がそのまま読み出されます。

●図……32　RR 15

bit 7
6
5
4
3
2
1
bit 0
Break/Abort IE
Tx Underrun/EOM IE
CTS IE
SYNC/Hunt IE
DCD IE
'0'
Zero Count IE
'0'
1：ゼロカウント割り込み許可
0：　〃　禁止
1：DCD変化割り込み許可
0：　〃　禁止
1：SYNC/Hunt状態変化による割り込み許可
0：　〃　禁止
1：CTS変化割り込み許可
0：　〃　禁止
1：Txアンダーラン/EOM発生による割り込み許可
0：　〃　禁止
1：Break/Abort状態検出による割り込み許可
0：　〃　禁止

キーボード/マウス

ユーザとの直接の接点となるのがキーボードとマウスです。X 68000 では，キーボードにたんなる文字入力のほか，ディスプレイやマウス制御機能も持たせ，ユーザインタフェースをトータルにサポートしています。

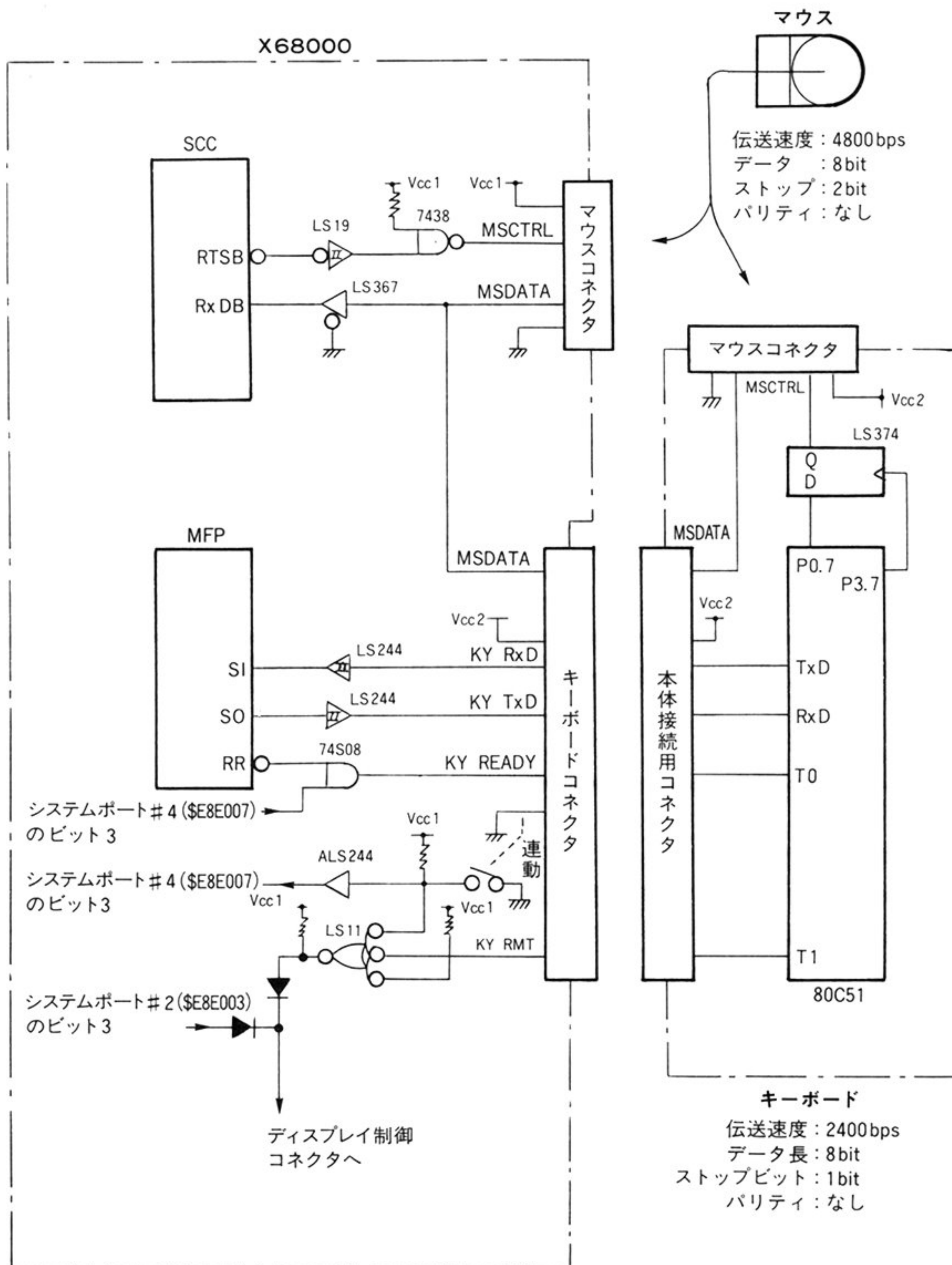
1 キーボード/マウスの概要

キーボードとマウスインタフェースのブロック図を354ページの図1に示します。X 68000では，キーボードとのデータ入出力をMFPのUSART(シリアルポート)，マウスからのデータ入力をSCCのBポートで行います。

キーボードとのデータ伝送は伝送速度2400 bps，データ長8ビット，ストップビット1ビット，パリティなしで，マウスとのデータ伝送は伝送速度4800 bps，データ長8ビット，ストップビット2ビット，パリティなしとなっています。

X 68000では，キーボードや本体から専用ディスプレイの電源ON/OFFやTVのチャンネルの切り替えなどの制御が行えるようになっていますが，この制御はキーボード内のCPU(ワンチップマイコン：80C51)やシステムポート#2で行うようにしています。キーボードの電源は本体の電源がOFFになっていても供給され続けるVCC 2からとっていますので，本体の電源がOFFであってもキーボード内のCPUは動作しており，キーボードによるTV制御が行えるようになっています。

また，X 68000では本体とキーボードの両方にマウスコネクタがついていますが，この両方

●図……1　キーボード/マウス系統ブロック図

のデータ線は電気的につながっています。ただし，マウスにデータ出力を要求するMSCTRL信号は，本体側はSCCのRTSB端子，キーボード側はキーボード上のCPUによって制御されるようになっています。キーボード側のMSCTRL信号の制御は，CPUへのコマンドによって行えるようになっています。

●2 キーボード/マウス関連ポート

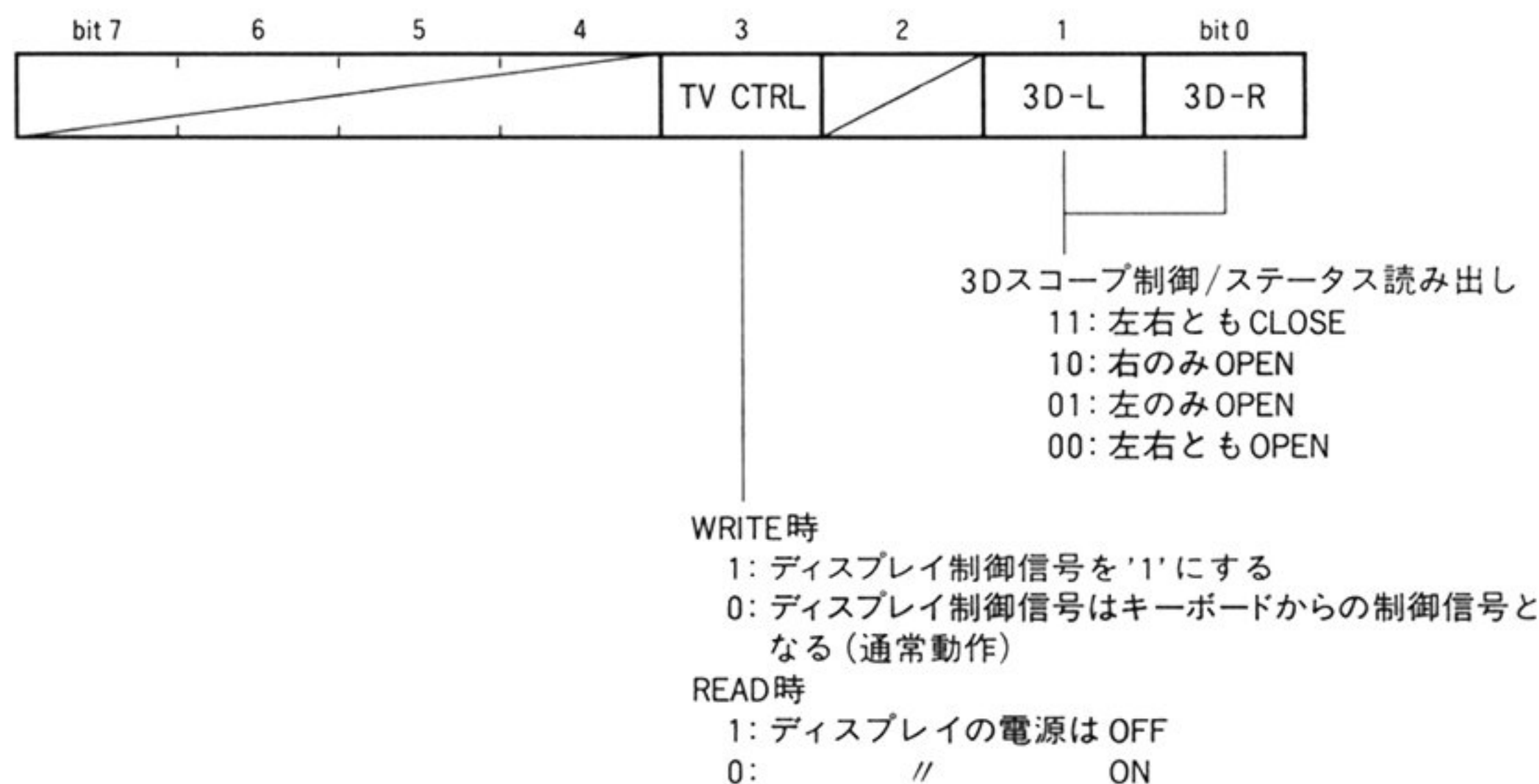
キーボードとマウスの制御に関連するI/Oポートを図2にまとめてみました。これらのうち，MFPとSCCについては，それぞれのデバイスの説明のページを参照してください。

●図……2　キーボード/マウス関連ポート

デバイス	アドレス	READ/WRITE	bit 7	6	5	4	3	2	1	bit 0	レジスタ
MFP	$E88027	R/W	SYNC								同期キャラクタレジスタ
	$E88029	R/W	CLK	WL1	WL0	ST1	ST0	PE	E/O		USARTコントロールレジスタ
	$E8802B	R/W	BF	OE	PE	FE	F/S or B	M/CIP	SS	RE	受信ステータスレジスタ
	$E8802D	R/W	BE	UE	AT	END	B	H	L	TE	送信ステータスレジスタ
	$E8802F	R/W									USARTデータレジスタ
I/O コントローラ	$E8E003	R/W					TV CTRL		3D L	3D R	システムポート＃2
	$E8E007	R/W					KEY CTRL	NMI RESET	HRL		システムポート＃4
SCC	$E98001	R/W									SCCコマンドポート
	$E98003	R/W									SCCデータポート

❷·1 システムポート＃2

システムポート#2（アドレス：$E8E003）のビット配置を356ページの図3に示します。ビット0，1はオプションの3Dスコープを制御するためのもので，ビット3がディスプレイに関係するビットです。ビット3は，書き込み時はディスプレイ制御信号，読み出し時はディス

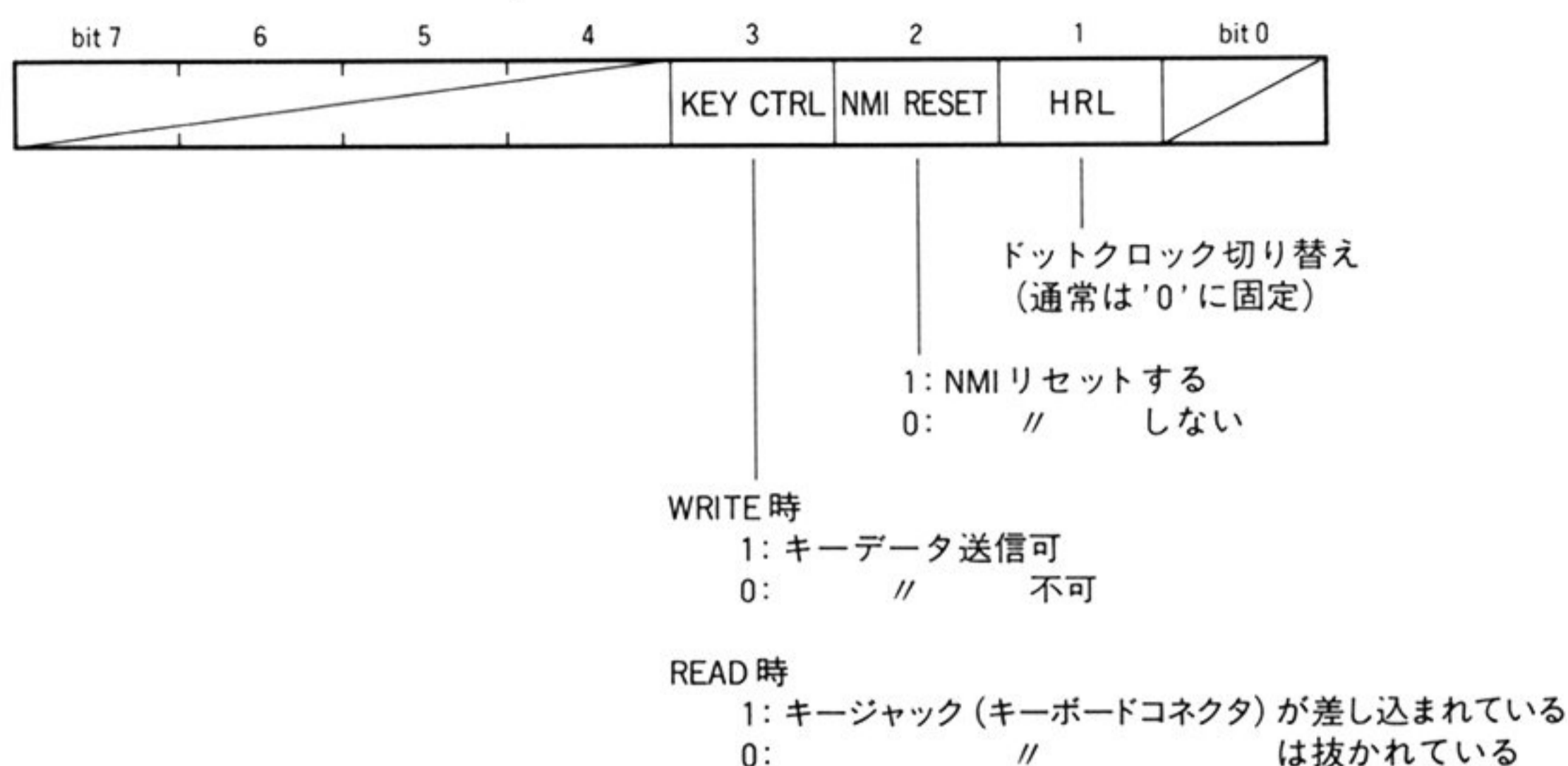
●図……3　システムポート＃2($E8E 003)

プレイの電源のON/OFFステータスとなります。

ビット3の出力はキーボードからのディスプレイ制御信号とダイオードORされています。通常，キーボードからのディスプレイ制御信号は'0'になっているため，このビットを'1'にするとディスプレイ制御信号は'1'に，'0'にすれば'0'になります。これによって，キーボードがない状態でもソフトウェアでディスプレイ制御を行うことができます。

このビットに'1'を書いたままにしておくと，ディスプレイ制御信号は'1'に固定されたままとなるため，キーボードからの制御が行えなくなります。さらにディスプレイ内部では，この信号とワイヤレス（赤外線）リモコンからの信号がORされるようになっているらしく，リモコンによる制御も行えなくなります。通常，このビットは'0'にするようにしてください。

❷·2　システムポート＃4

システムポート＃4（アドレス：$E8E007）のビット配置を図4に示します。ビット3でキーボードに対してキーデータの送出が可能か否かを示します。通常，キーボードからデータが送られてくると，MFPはRR（Receiver Ready）信号を'1'（Lowレベル）にし，CPUがデータを読み取ると'0'（Highレベル）に復帰させます。キーボードはこの信号をチェックし，'0'になっているときだけキーデータを送るようにすることで，CPUがデータを引き取らないうちに次のデータを送ってしまうようなことを避けているわけです。

●図……4　システムポート#4($E8E007)

システムポート#4のビット3に'0'を書き込むと，この信号が強制的に'1'（Lowレベル）にされ，キーボードはデータ送出が行えなくなります。キーボードはキーデータの送出を行った後で次のキースキャン（キーが押されたのか，離されたのかをチェックする動作）を行いますので，この状態ではキーボードからのディスプレイ制御も行えなくなります。

このビットは読み出し時にはキーボードの挿抜ステータスとして機能します。キーボードが挿し込まれていると'1'に，抜かれていると'0'になります。

●3 キーボードからの入力データ

キーボードから本体に送られてくるキーデータのフォーマットを358ページの図5に示します。キーデータはキーが押されたときと離されたときのいずれの場合も通知されます。下位7ビットで変化があったキーのキーコードが示され，ビット7で，そのキーが押されたのか離されたのかを示します。X 68000の各キーの配置とキーコードの対応は358ページの図6のようになっています。

●図……5　キーデータ

bit 7　6　5　4　3　2　1　bit 0

Make/Break　Key Code

キーコード

1: キーが離された
0: 〃 押された

●図……6　キー配列とキーコード

キーコードは16進数

●4 キーボードへの出力データ

X 68000本体からキーボードへ与えるコマンドの一覧を図7に示します。X 68000では，キーボード中のCPUがディスプレイ制御信号を発生したり，マウスコントロール信号（MS CTRL）の制御を行うようにしているため，それらの機能をサポートするためのコマンドが多

●図……7　キーボードへの制御コマンド

データ								機　能
bit 7	6	5	4	3	2	1	bit 0	
'0'	'0'		TV CTRL Code					専用ディスプレイ(ディスプレイTV)制御
'0'	'1'	'0'	'0'	'0'			MS CTRL	キーボードのマウスコネクタのMSCTRL信号制御
'0'	'1'	'0'	'0'	'1'			KEY EN	キーデータ送出許可/禁止
'0'	'1'	'0'	'1'	'0'	'0'		X68K/X1	キー操作によるディスプレイ制御モード選択
'0'	'1'	'0'	'1'	'0'	'1'	BRIGHT		キーボード上のLEDの明るさ選択
'0'	'1'	'0'	'1'	'1'	'0'		CTRL EN	本体からのディスプレイ制御有効/無効
'0'	'1'	'0'	'1'	'1'	'1'		OPT2 EN	OPT2キーによるディスプレイ制御許可/禁止
'0'	'1'	'1'	'0'	REP. DELAY				キーが押されてからリピートが始まるまでの時間設定
'0'	'1'	'1'	'1'	REP. TIME				リピート間隔設定
'1'	全角	ひらがな	INS	CAPS	コード入力	ローマ字	かな	キーボード上のLEDの点灯/消灯制御

くなっています。

❹・1　ディスプレイコントロール

X 68000では，キーボードの操作によるディスプレイ制御だけでなく，本体からキーボードのCPUに対してディスプレイ制御信号の発生を要求することができるようにしています。このためのコマンドの一覧を360ページの図8に示します。

このコマンドでは，電源のON/OFFや，ノーマルコントラストでのスーパーインポーズなど，キーボードからの操作ではできないものも含まれています。とくにノーマルコントラストでのスーパーインポーズは，通常のスーパーインポーズ時よりもTV画面が明るくなりますので，スーパーインポーズのときは，このモードを利用したほうがよいでしょう。

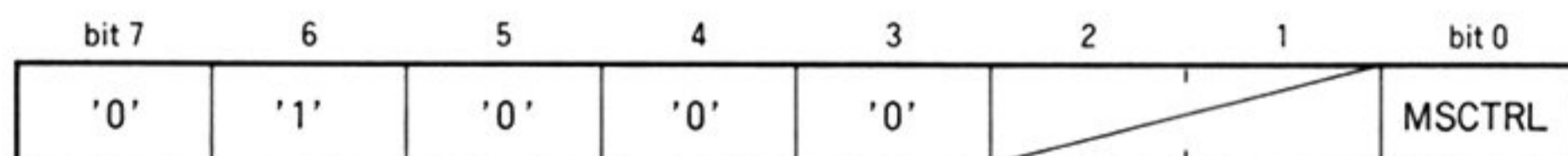
❹・2　マウスコントロール信号制御

コマンドのフォーマットを360ページの図9に示します。ビット0でキーボードについているマウスコネクタのMSCTRL信号の状態を選択します。マウスは，MSCTRLがHighから

●図……8　ディスプレイコントロールコマンド一覧

コントロールコード	SHIFTキーと同時に押すキー	名　称	機　　能
$00	——	——	（無　効）
$01	↑	Vol. up	音量（ボリューム）up
$02	↓	Vol. down	〃　down
$03	,	Vol. normal	〃　ノーマル
$04	CLR	Call	チャンネルコール
$05	（該当キーなし）	CS down	テレビ画面（初期化，リセット）
$06	0	Mute	音声ミュート
$07	——	CH16	（無　効）
$08	.	BR up	テレビ/コンピュータ画面切り替え（トグル）
$09	=	BR down	テレビ/外部入力切り替え（トグル）
$0A	（該当キーなし）	BR ½	コントラストノーマル
$0B	→	CH up	チャンネルup
$0C	←	CH down	チャンネルdown
$0D	——	——	（無　効）
$0E	（該当キーなし）	Power ON/OFF	電源ON/OFF（トグル）
$0F	+	CS ½	スーパーインポーズON/OFF（トグル），コントラストダウン
$10	テンキーの1	CH 1	チャンネル 1
$11	〃　2	CH 2	2
$12	〃　3	CH 3	3
$13	〃　4	CH 4	4
$14	〃　5	CH 5	5
$15	〃　6	CH 6	6
$16	〃　7	CH 7	7
$17	〃　8	CH 8	8
$18	〃　9	CH 9	9
$19	〃　/	CH 10	10
$1A	〃　*	CH 11	11
$1B	〃　−	CH 12	12
$1C	〃　=	CH 13	テレビ画面
$1D	〃　.	CH 14	コンピュータ画面
$1E	〃　+	CH 15	スーパーインポーズON/OFF（トグル），コントラストダウン
$1F	（該当キーなし）	——	〃　，コントラストノーマル

＊$1C～$1Fは，X1コンパチモード時の対応キーを表記

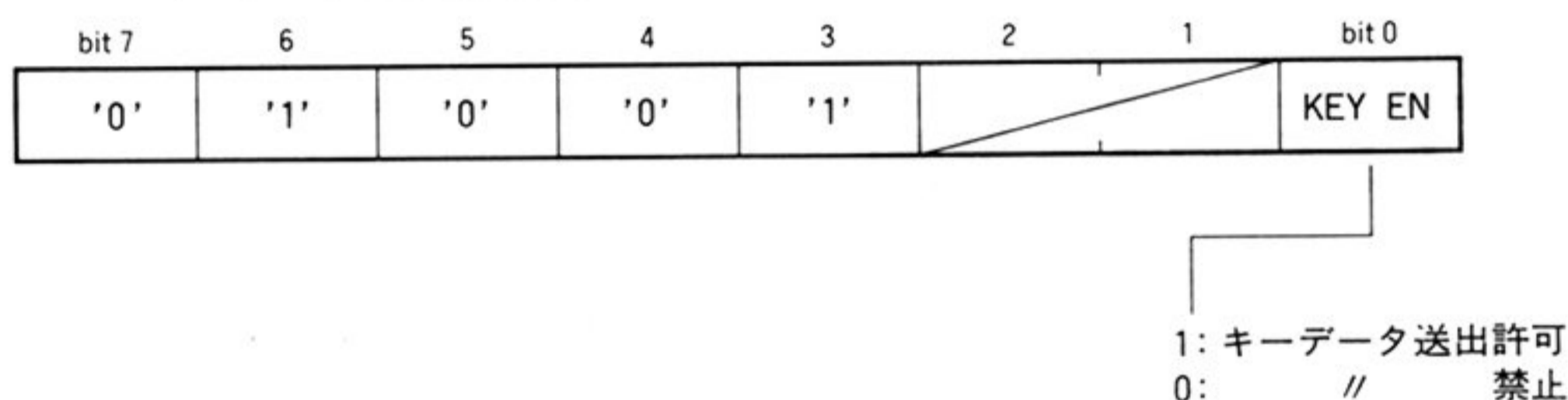
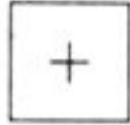
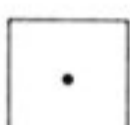
●図……9　マウスコントロール信号制御

Lowになったのをとらえてデータの送出を開始します。

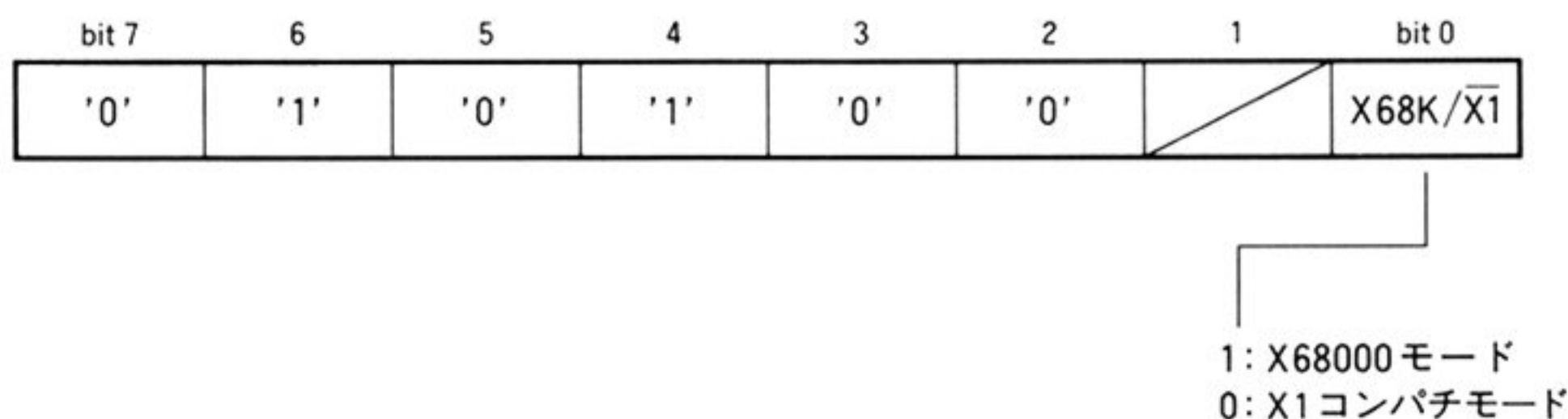
❹·3 キーデータ送出許可/禁止

CPU がキーデータを引き取らなかったり，システムポートでキーボードにキーデータの送出を禁止したりすると，キーボードはキーデータの送出が行えるようになるまでキースキャンを停止してしまうため，キーボードによるディスプレイ制御も行えなくなります。

このような状態を避けるため，キーボードに対してキーデータを送出せずにキースキャン動作を行わせるようにするのが，このコマンドです。このコマンドでキーデータの送出を禁止すると，キーボードの CPU はキーデータを本体に送出するのをやめますが，ディスプレイ制御信号の発生は行いますので，キーボードによるディスプレイ制御は通常どおり行うことができます。

コマンドフォーマットは図 10 のようになっています。最下位ビットを '0' にするとキーデータ送出が禁止され，'1' にすると通常の動作モードになります。

●図……10 キーデータ送出許可/禁止

❹·4 ディスプレイコントロールキーモード

キー操作によるディスプレイ制御を，X 1 とコンパチブルなモードにするか否かを選択します。コマンドのフォーマットは図 11 のようになっています。

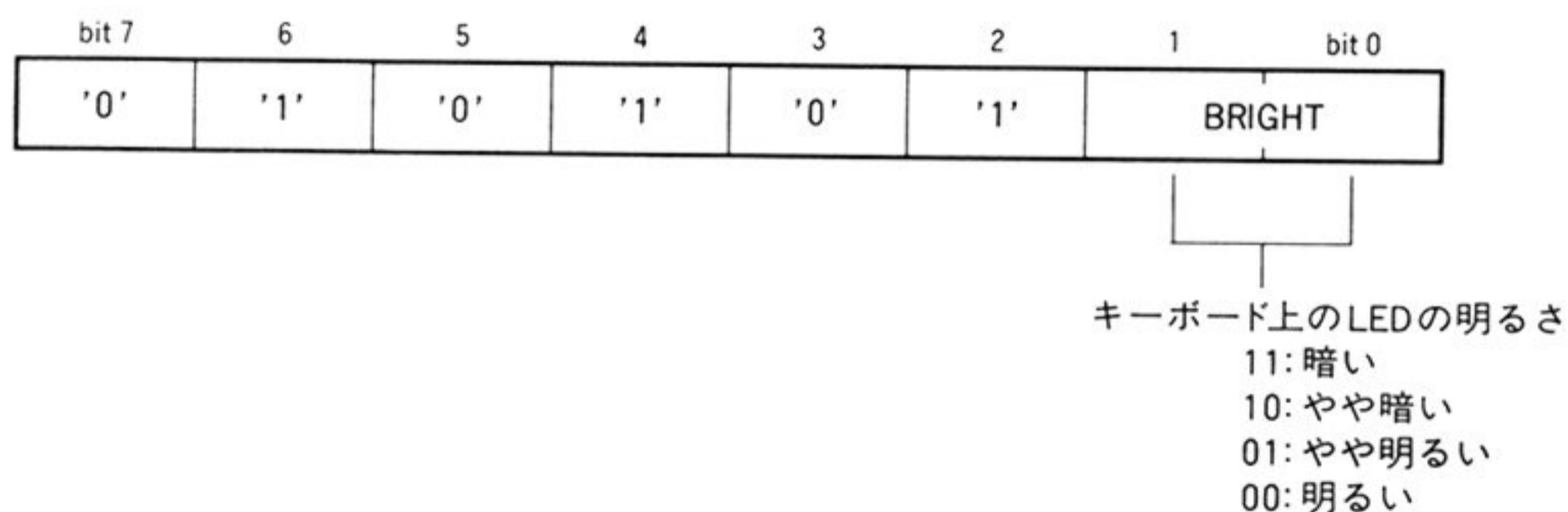
●図……11 ディスプレイコントロールキーモード

通常のモード(X 68000 モードと呼ぶことにします)とX 1 コンパチモードの違いを図 12 に示します。X 68000 モードでは，スーパーインポーズや入力の切り替えが，データを送るたびにトグル(交互に切り替わる)しますが，X 1 コンパチモードでは，キー入力によってスーパーインポーズ，TV，コンピュータの選択になります。

●図……12 TV コントロール操作

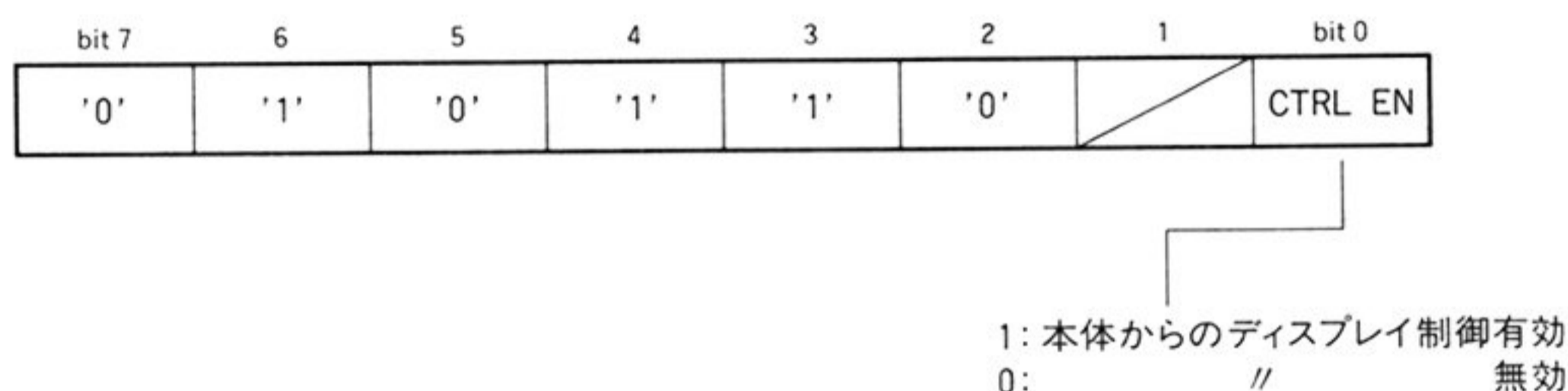
SHIFTキーと同時に押すキー	X68000モード	X1コンパチモード
+	スーパーインポーズ ON/OFF (トグル)	スーパーインポーズ
=	TV/外部入力切り替え (トグル)	TV
·	TV/コンピュータ切り替え (トグル)	コンピュータ

❹·5 LED明るさ選択

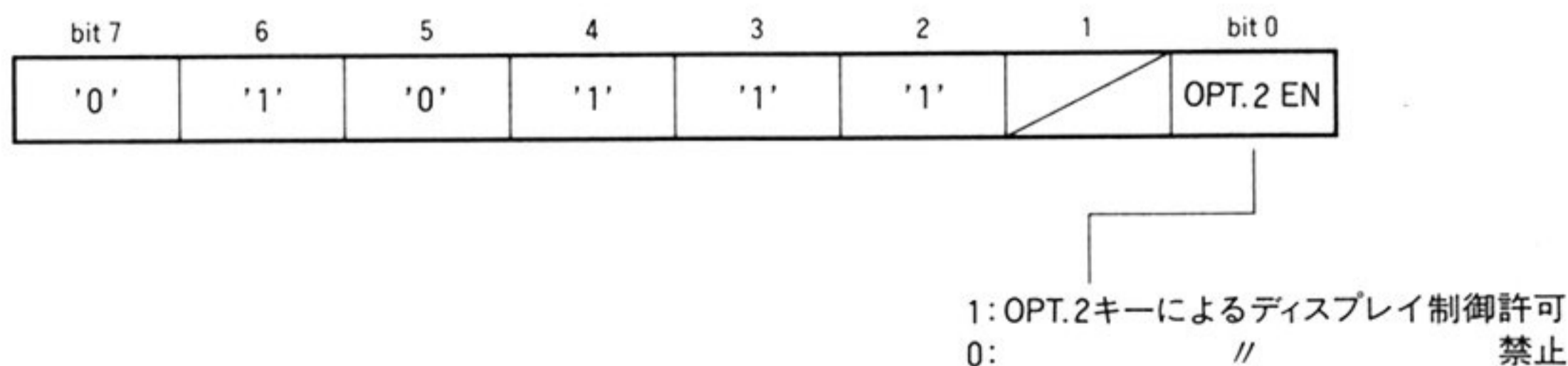
キーボード上の LED の明るさの選択を行います。コマンドフォーマットは図13のようになっています。下位 2 ビットで LED の明るさを 4 段階に調整できます。

●図……13 LED 明るさ選択

❹·6 本体からのディスプレイ制御の有効/無効選択

本体からキーボードに要求するディスプレイ制御コマンドを受け付けるか否かを選択します。コマンドフォーマットは図14のようになっており，最下位ビット（ビット0）を'0'にすると，制御コマンドが無効になります。

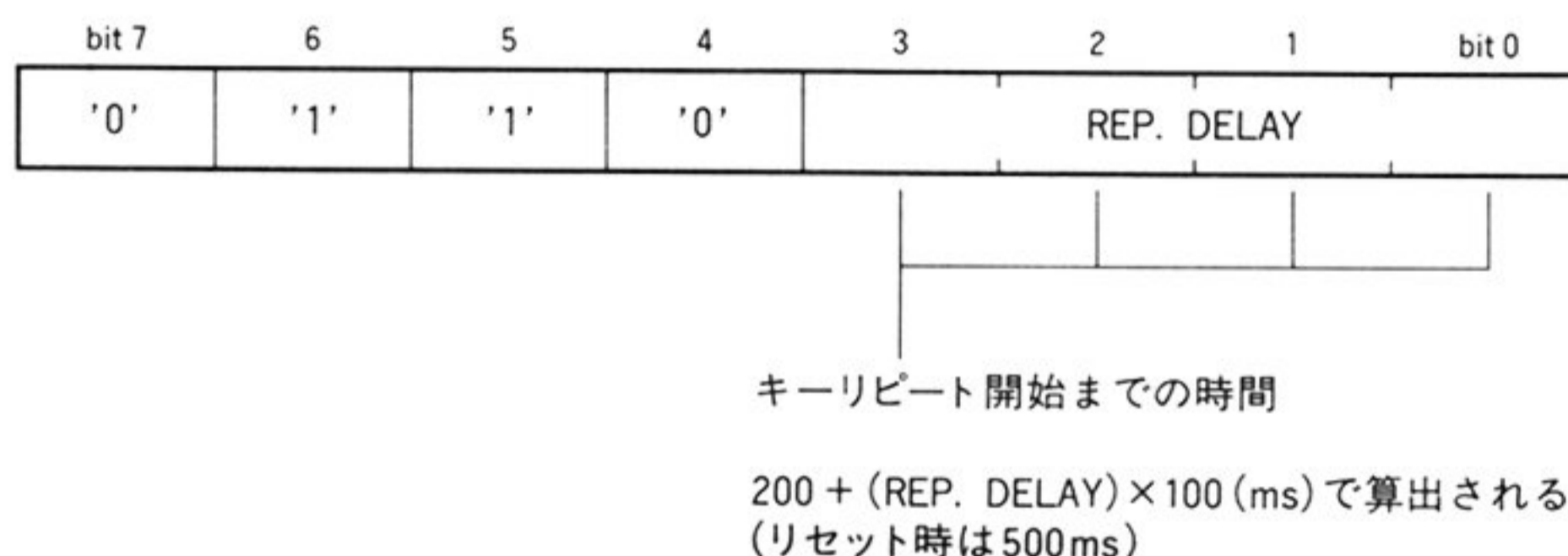
●図……14　本体からのディスプレイ制御有効/無効

❹·7 OPT.2キーによるディスプレイ制御許可/禁止

コマンドのフォーマットは図15のようになっています。キー操作でのディスプレイ制御は通常SHIFTキーを用いますが，OPT.2をSHIFTキーの代用として使うこともできるようになっています。このコマンドは，このOPT.2キーによるディスプレイ制御を許可するか，禁止するかを選択するものです。

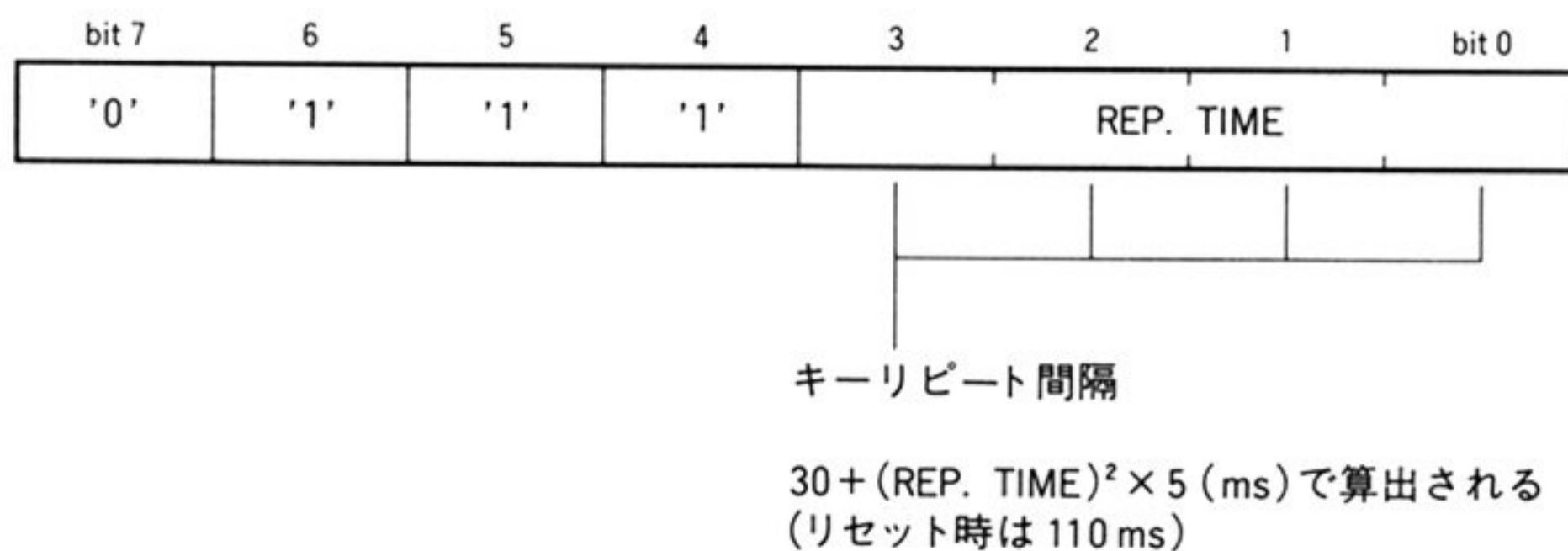
●図……15　OPT.2 キーによるディスプレイ制御

❹·8 キーリピート開始時間設定

コマンドのフォーマットは図16のようになっています。キーを押し続けたとき，キーリピートが開始されるまでの時間を設定します。

下位4ビットによって，リピート開始までの時間を200 msから1700 msまで100 ms単位で設定することができます。キーボードがリセットされたときは，この時間は500 msに初期設定されます。

●図…… 16　キーリピート開始時間設定

❹·9 キーリピート間隔設定

コマンドのフォーマットは図17のようになっています。キーリピートの間隔を30 ms～1155 msの間で設定することができます。キーボードがリセットされたときは，この間隔は110 msに設定されます。

●図…… 17　キーリピート間隔コマンド

❹·10 キーボードLED制御

キーボード上にあるLED付きのキーのLED点灯/消灯を制御します。コマンドフォーマットは図18のようになっています。下位7ビットがそれぞれのキーに対応しており，ビットが'1'のとき消灯，'0'のとき点灯します。

●図……18 キーボードLED制御

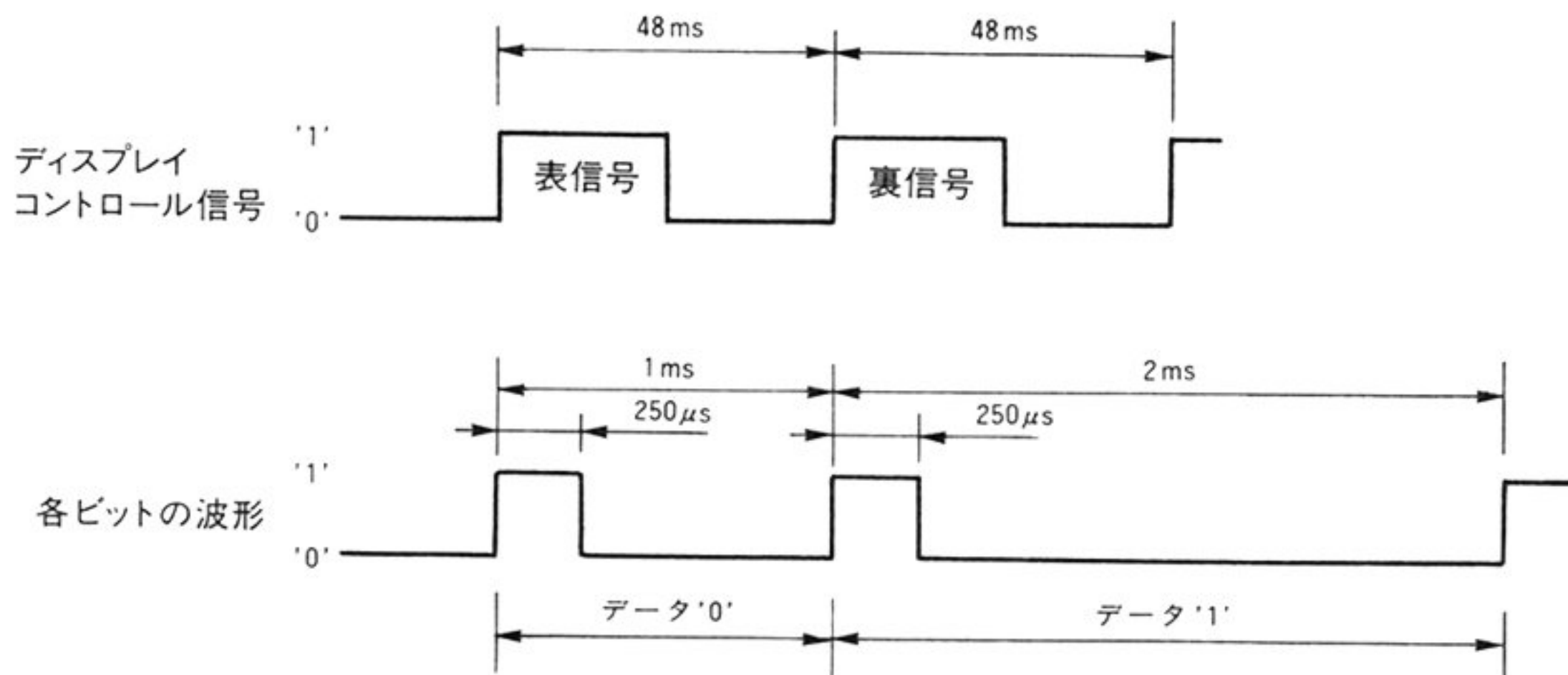
●5 ディスプレイ制御信号

キーボードからのディスプレイ制御信号がきていないときや，キーボードコネクタが抜けているときには，システムポート#2のビット3を制御してディスプレイ制御信号をつくり出せば，本体だけでもディスプレイ制御を行うことができます。

ディスプレイ制御信号のフォーマットは366ページの図19のようになっています。1回のデータ送出は48 msの間隔をおいて表信号と裏信号を送ることで行われます。表信号と裏信号のデータの内容を図20に示します。裏信号は基本的には表信号の反転データであり，ディスプレイ側ではこの両方が正しく受け取れたのを確認してからコマンドを実行することでノイズ等による誤動作を防いでいるわけです。

ディスプレイ制御信号の各ビットの情報は，単純にデータの1/0を信号の1/0に対応させているのではなく，図に示したように，250 μs幅のパルスの後，次のパルスまで何msの間隔をあけるかということで表すようにしています。

これらの方法はワイヤレスリモコンで行われている方法です。ディスプレイ内部では，ワイ

●図……19　ディスプレイコントロール信号

●図……20　ディスプレイへの送出データ

信　号	ディスプレイに送出するデータ(左側のビットから順に送出される)											
	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	K
表信号	'0'	'0'	'0'	ディスプレイコントロールコード					'0'	'0'	'0'	'0'
裏信号	'0'	'0'	'0'	ディスプレイコントロールコードの反転					'1'	'1'	'1'	'1'

ヤレスリモコンから受け取った信号と本体から送られてくる信号が単純にORされてリモコンデータとなっているようです。

6　キーボードの特殊機能

本来の用途とはあまり関係ありませんが，キーボードが持っているおまけ的な機能を紹介しておきましょう。

6·1　LEDの明るさ指定

キーボードをリセットするとき(キーボードを抜き挿しするとき)にLEDの明るさ選択が行

えます。

・何も押さないで立ち上げたとき　　：明るい
・XF 3を押しながら立ち上げたとき：やや明るい
・XF 4を押しながら立ち上げたとき：やや暗い
・XF 5を押しながら立ち上げたとき：暗い

❻・2 LEDチェック

F 1，F 2，F 3の3つのキーを同時に押しながらキーボードをリセットすると，LEDが点滅を繰り返します。この状態では，キーボードからの入力などはまったく行えませんので，使用するときはキーを押さない状態で再度キーボードリセットを行ってください。

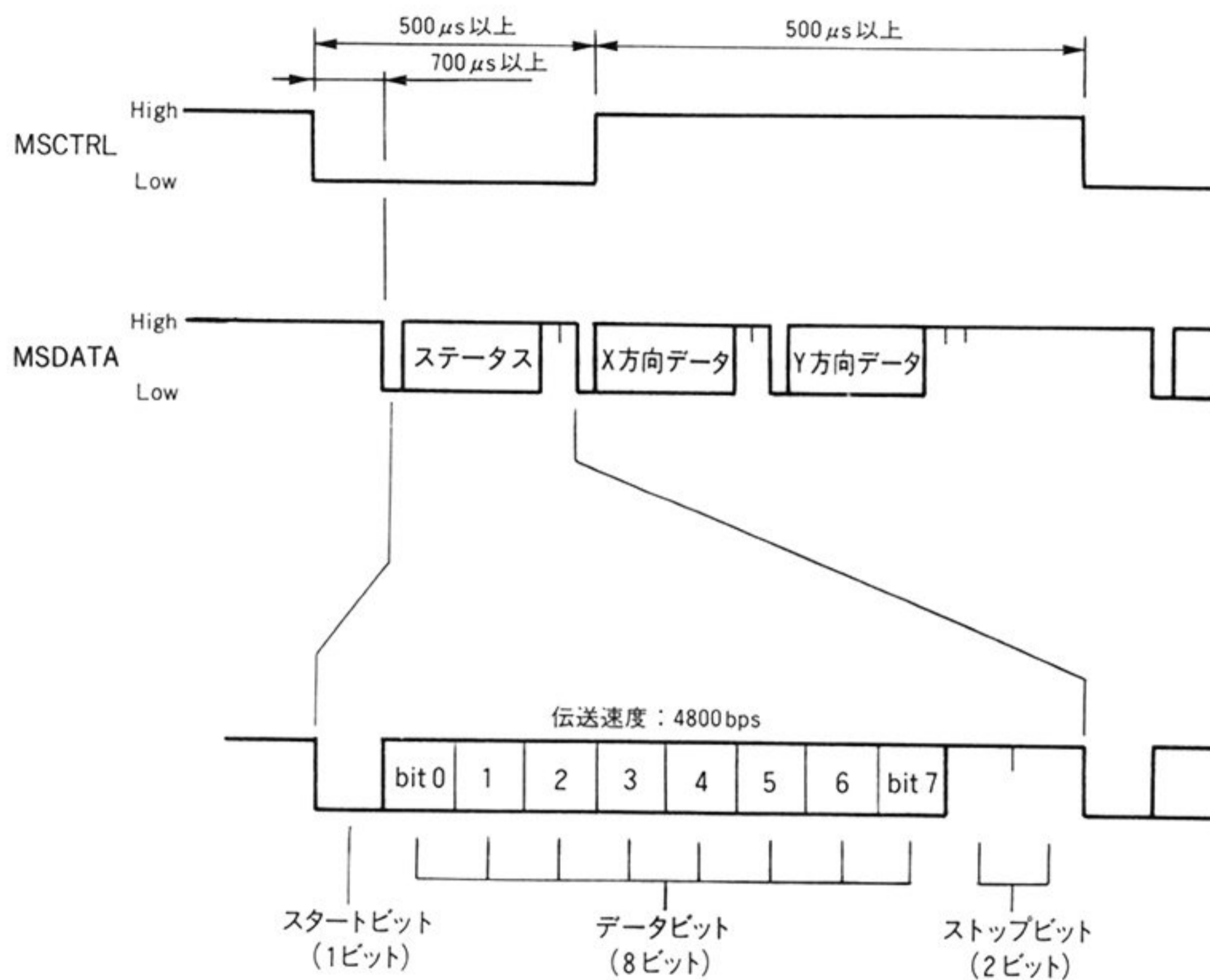
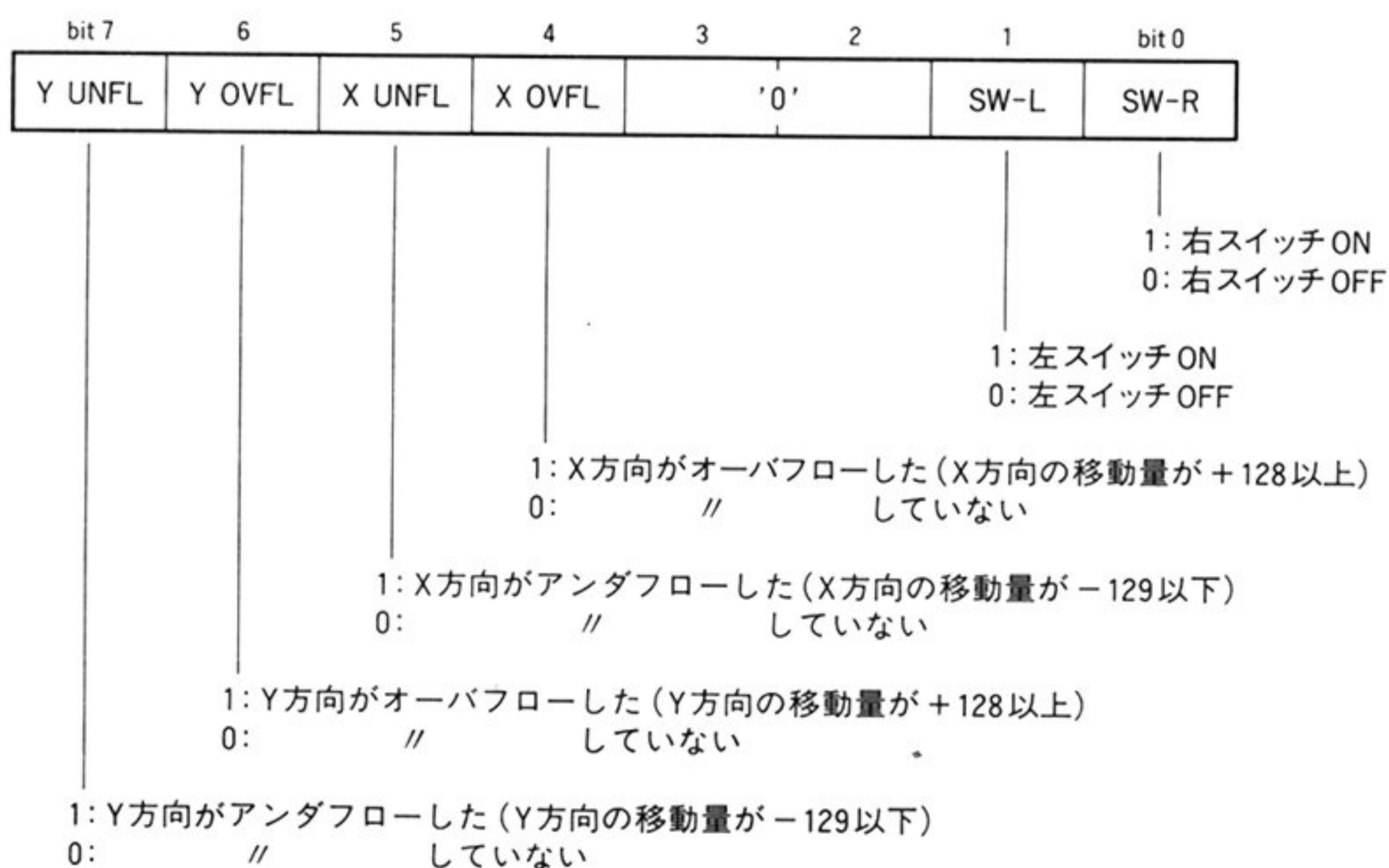
●7 マウス制御

マウスは，マウス制御信号（信号名：MSCTRL）がHighからLowに変化すると，ステータス，X方向データ，Y方向データの3バイトデータを送ってきます。マウス制御タイミングの規定などを368ページの図21に示します。

マウスデータ信号（信号名：MSDATA）は本体のマウスコネクタとキーボードからのものが単純に接続されているだけですが，MSCTRL信号は本体側がSCC，キーボード側はキーボード内のCPUで制御されるようになっているため，マウスがどちらに接続されていてもよいようにするためには，SCCとキーボードの両方でMSCTRL信号を操作する必要があります。

マウスから送られてくるX方向，Y方向のデータは符号付き2進数で，$80が−128，$7Fが+127を示します。このデータは前回データを送りはじめた時点からの相対的な移動量を示します。

マウスデータの先頭バイトであるステータスデータのフォーマットは368ページの図22のようになっています。上位4ビットはそれぞれY方向，X方向でアンダフロー（移動量が−129以下になってしまい，移動量データでは表現しきれなくなった）やオーバフロー（移動量が+128以上になったため，移動量データでは表現しきれなくなった）を検出すると，該当するビッ

●図……21　マウスのデータ転送タイミング

●図……22　マウスのステータスデータ

トが '1' になります。

ビット 1 とビット 0 は，マウスの左右のスイッチの状態を示します。'1' のときスイッチが押されていることを，'0' のとき離れていることを示します。

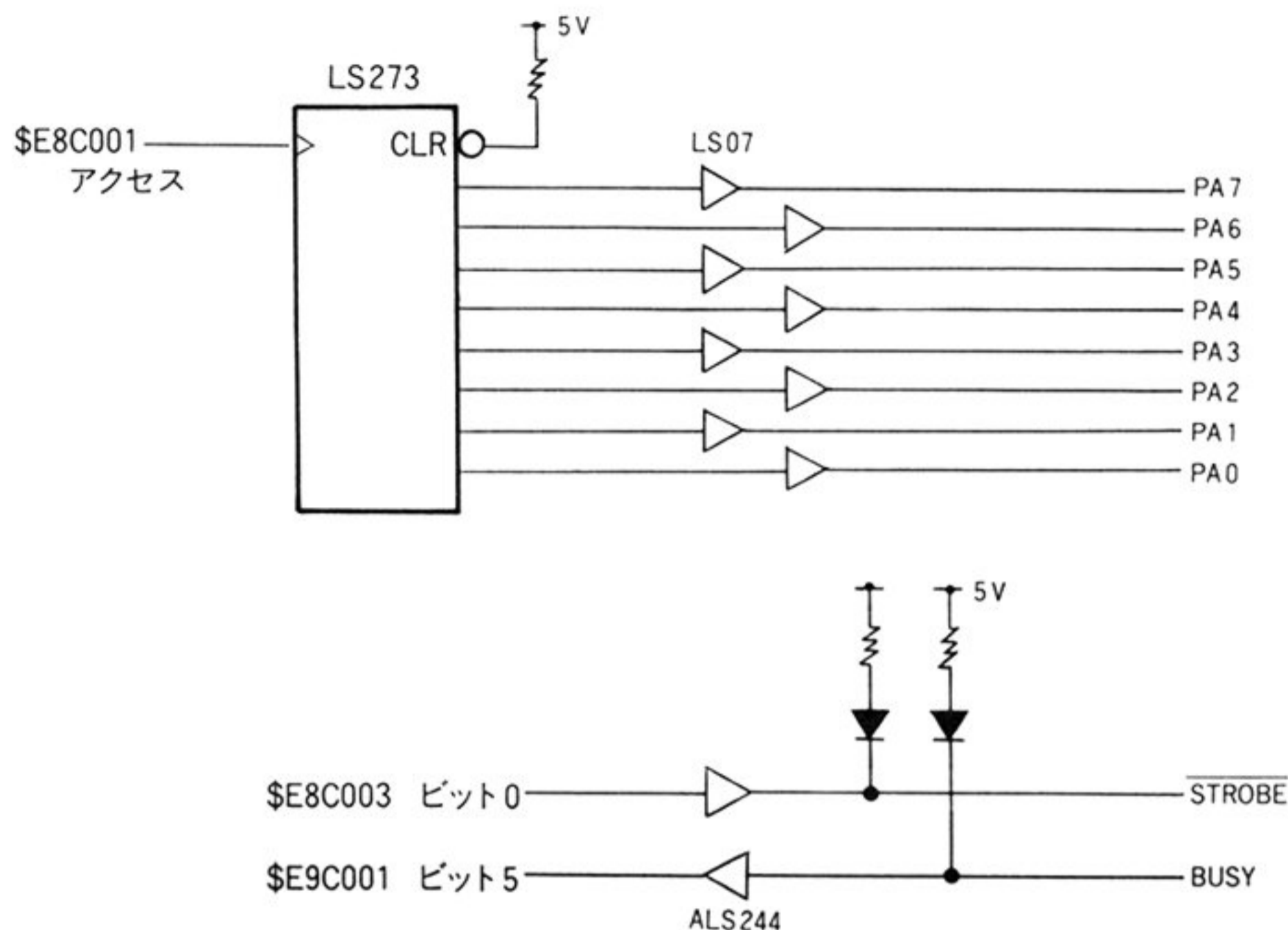
プリンタ

14 ピンという，小さなコネクタにまとめられたプリンタインタフェースはX1とコンパチブルなものとなっています。ここでは，プリンタインタフェースの構成と，インタフェース信号について説明します。

1 プリンタインタフェースの概要

X 68000では，プリンタとしてセントロニクスインタフェース準拠のものが接続できるようになっています。X 68000のプリンタインタフェースのブロック図を372ページの図1に示します。PA 0～PA 7は8ビットのデータライン，STROBEはプリンタに対してデータの引き取りを要求するもの，BUSYはプリンタが次のデータを受け付ける準備ができているかどうかを示す信号です。X 68000ではBUSY信号をI/Oコントローラに入力し，ビジー状態からレディ状態への変化時（次のデータ転送が行えるようになったとき）に割り込みを発生することもできるようになっています。

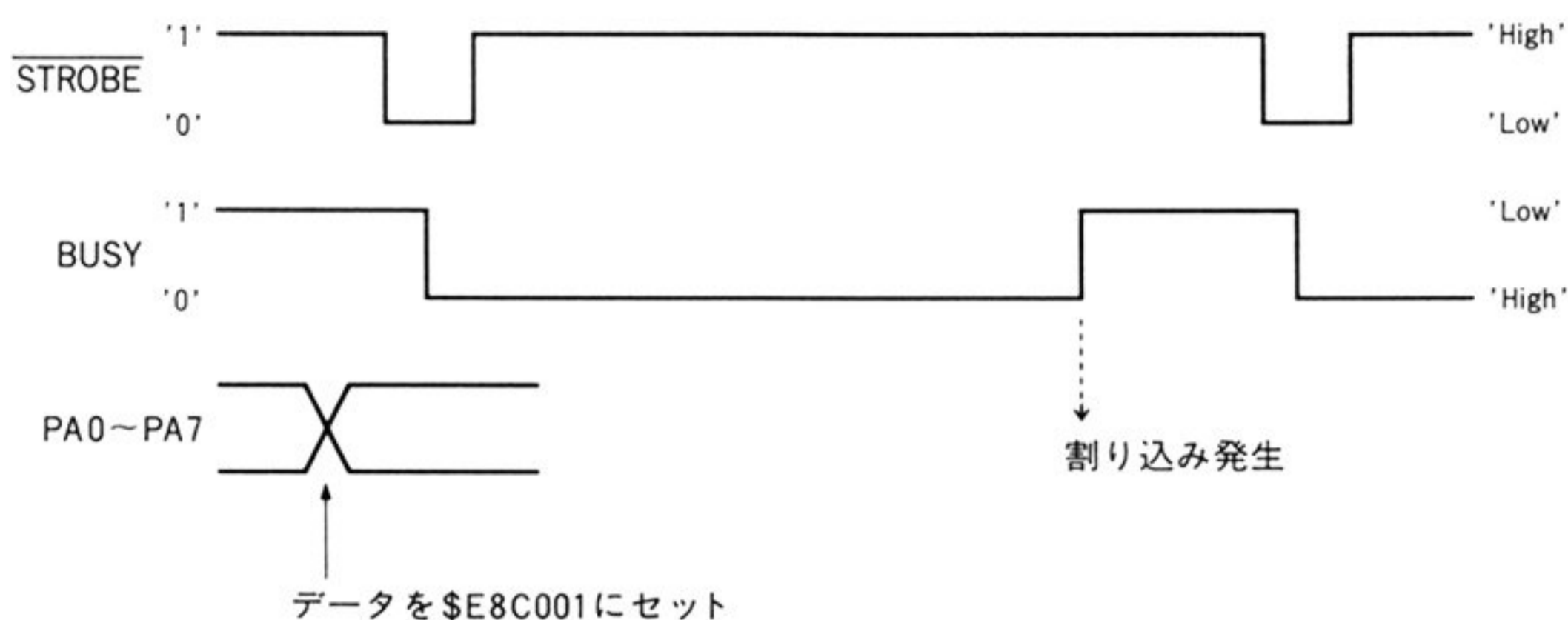
セントロニクスインタフェースを採用しているプリンタは，ほとんどがプリンタ上のSELECTスイッチや紙切れ状態，データ引き取り完了パルスなどを個別の信号線で出力しているのですが，X 68000ではこれらの信号は使用していません。X 68000側ではBUSY信号によってビジー（データ引き取り不可）か，レディ（データ引き取り可）かをチェックし，一定時間たってもビジーのままであれば，タイムアウトエラーとして処理することになります。

●図……1　プリンタインタフェースブロック図

❶·1　プリンタ制御タイミング

プリンタ制御のタイミング例を図2に示します。STROBE信号は'1'，プリンタからのBUSY信号は'1'になっているものとします。なお，ここでの'1'や'0'はポートに書き込んだり，ポートから読み出されるデータを指します。

まず，BUSY信号が'1'になっている（プリンタがレディ状態である）ことを確認して，PA 0～PA 7にプリンタに送りたいデータをセットします。次にSTROBE信号を'0'にすると，プリンタがデータを引き取るため，BUSYが'0'になりますので，これを見てSTROBEを'1'に復帰させます。プリンタ側でデータが引き取られ，次のデータ引き取りの準備ができると，BUSYが'1'に復帰します（必要ならば，この時点で割り込みを発生させることもできます）。X 68000側はBUSYが'0'に復帰したのを見て，次のデータの送出を行うわけです。

BUSY信号が'1'になっている期間はプリンタ側の都合で決まるものであり，どの程度になるかはわかりません。また，実際にはBUSY信号をチェックせずにSTROBEを'0'，'1'と連続して変化させて，BUSYが'1'になるのを待つという方法がよくとられているようです。

●図……2　プリンタ制御タイミング例

*：I/Oコントローラのポート($E9C001)で読み出される状態

2 プリンタ関連ポート

プリンタインタフェースに関連するポートの一覧を図3に示します。プリンタに送るデータは$E8C001にセットし，STROBE信号を$E8C003で制御します。

プリンタからのBUSY信号がビジー状態からレディ状態へ変化したときに割り込みを発生させることができるほか，$E9C001の上位ビットでステータスとして読み出すこともできるようになっています。

●図……3　プリンタ関連ポート

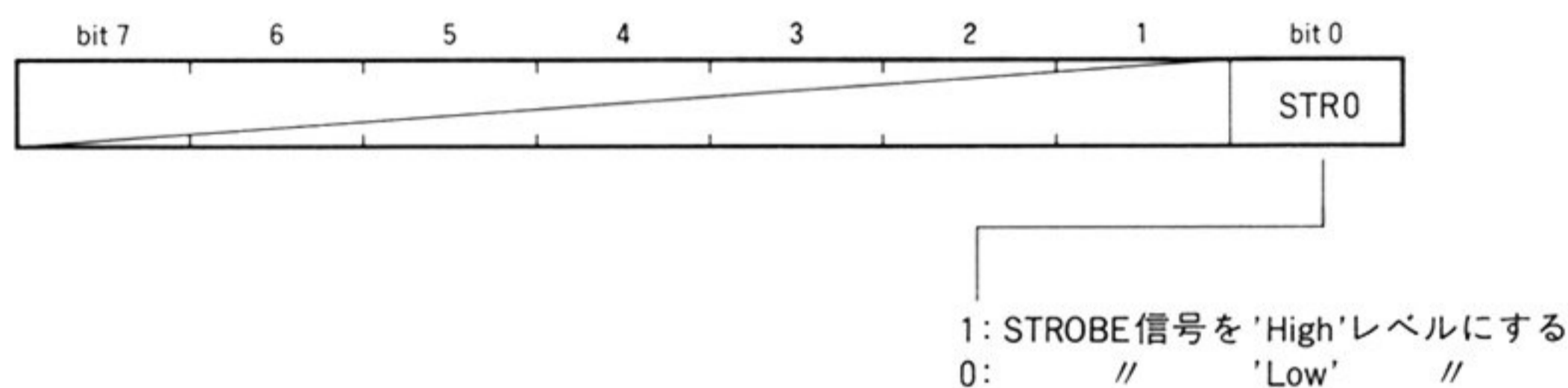
アドレス	READ/WRITE	bit 7	6	5	4	3	2	1	bit0	備考
$E8C001	W									プリンタデータ
$E8C003	W								STRO	プリンタストローブ
$E9C001	R	FDC INT	FDD INT	PRT INT	HDD INT	HDDI EN	FDCI EN	FDDI EN	PRTI EN	割り込みステータス
	W					HDDI EN	FDCI EN	FDDI EN	PRTI EN	割り込みマスク
$E9C003	W	Vect						DEVICE		割り込みベクタ

❷·1 プリンタデータポート

プリンタに送出するデータをセットします。STROBEを操作する前に，このポートにデータをセットしておくようにしてください。

❷·2 プリンタストローブポート

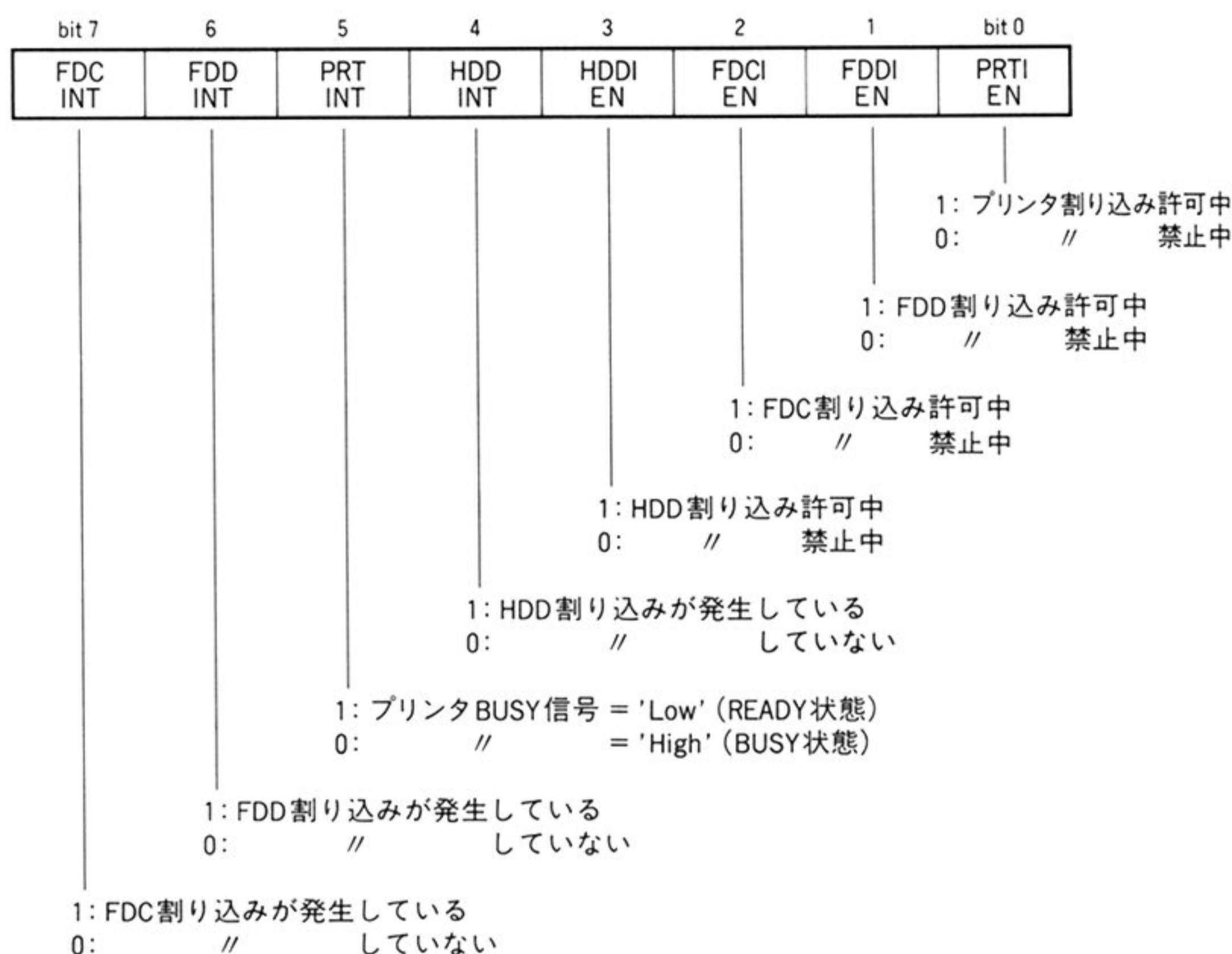
プリンタストローブポートのビット配置を図4に示します。最下位ビットがSTROBE信号の制御ビットとなっており，'1'にするとSTROBE信号がHighレベルに，'0'にするとLowレベルになります。

●図……4　プリンタストローブレジスタ($E8C003)

❷·3 割り込み信号ステータス

割り込み信号ステータスポートのビット配置を図5に示します。上位4ビットは各割り込み要因が発生しているか否かを示すビット，下位4ビットは割り込みマスクレジスタに書き込まれた値がそのまま読み出されます。

このうち，ビット5がプリンタの割り込み要求状態，すなわち，BUSY信号の状態を示し，ビット0がプリンタからの割り込みがマスクされているか否かを示しています。

プリンタの割り込みは，BUSY信号が'0'から'1'に変化したときに発生します。割り込みマスクレジスタによってプリンタ割り込みの発生が禁止されていても，ビット5にはBUSY信号の状態が反映されますので，これを使って割り込みを使用せずにプリンタの制御を行うこともできます。

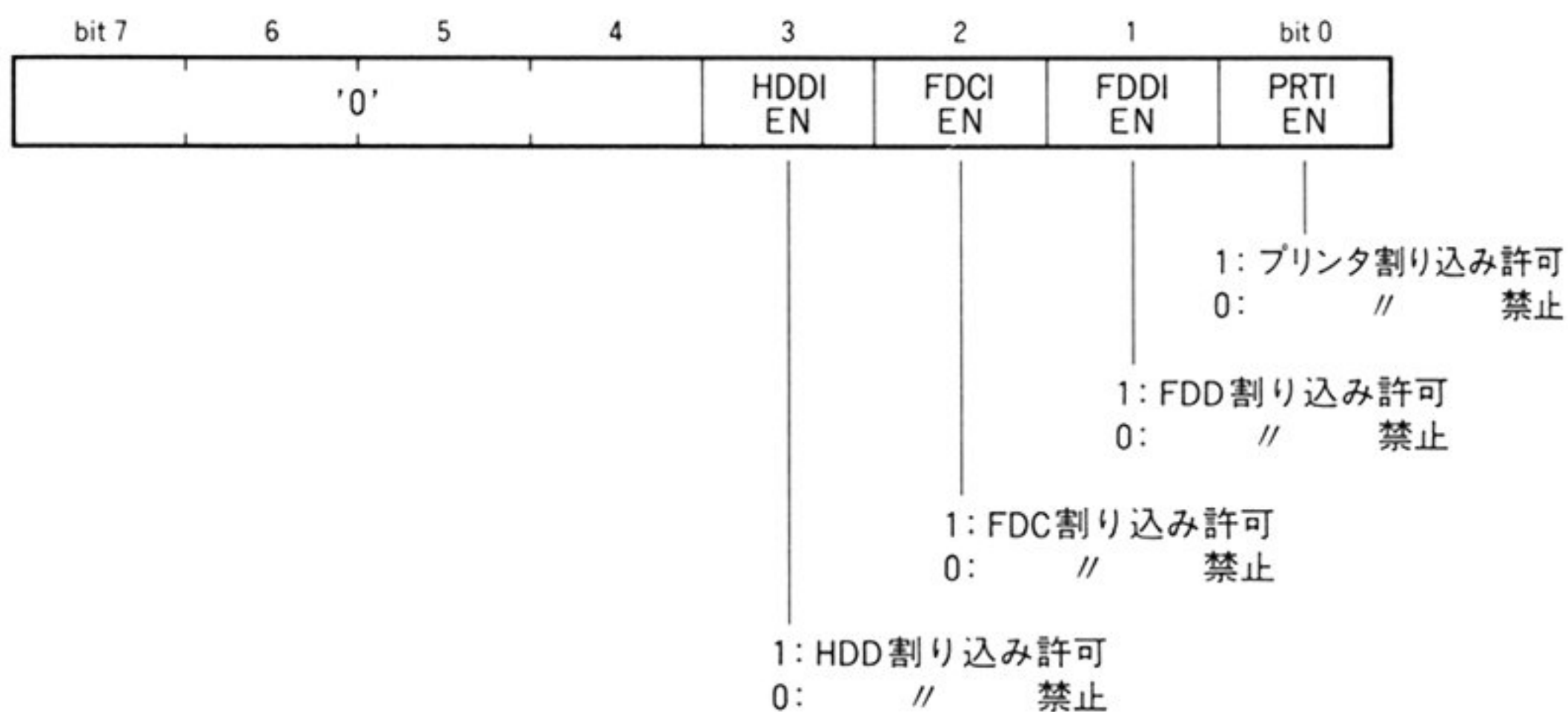
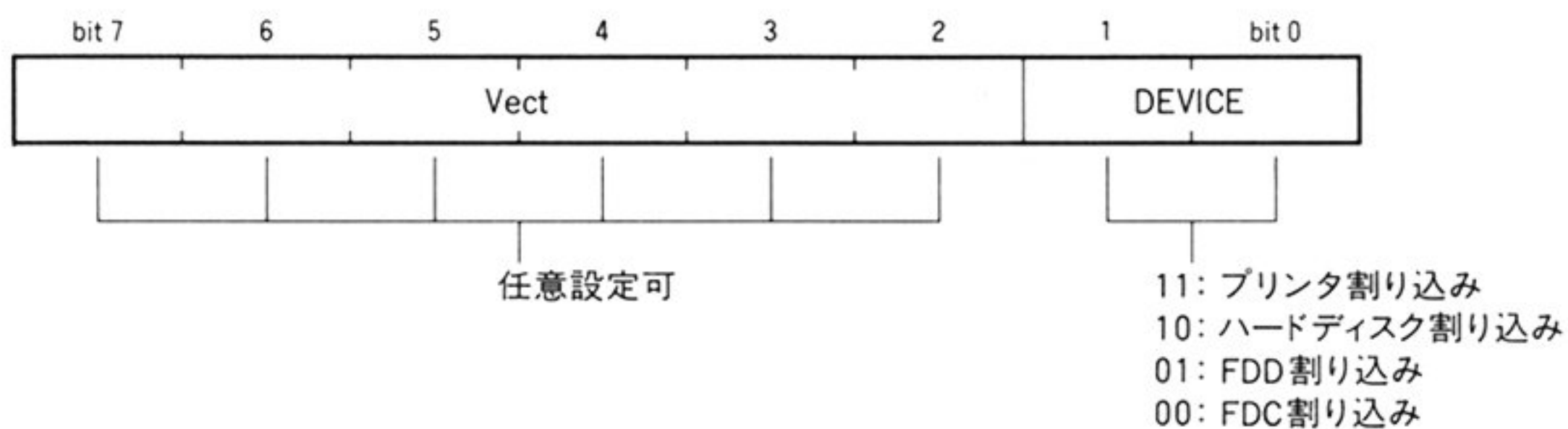
●図……5　割り込み信号ステータス($E9C001)

❷·4 割り込みマスク

I/O コントローラが管理している各割り込み要因ごとに，CPU への割り込み要求を行うか否かを決めるレジスタです。このレジスタのビット配置を 376 ページの図 6 に示します。

プリンタの割り込み制御はビット 0 で行います。このビットが '1' になっているとプリンタ割り込みの発生が許可に，'0' になっていれば禁止になります。

❷·5 割り込みベクタレジスタ

I/O コントローラの割り込みベクタ設定レジスタのビット配置は 376 ページの図 7 のようになっています。このレジスタは，割り込み発生時に CPU に与えるベクタ番号を設定します。上位 6 ビットは任意に設定可能で，下位 2 ビットは割り込み要因によって自動的に変化するため，CPU からの設定は無効になります。プリンタからの割り込みが発生したときは，下位 2 ビ

●図……6　割り込みマスク($E9C001)

●図……7　割り込みベクタ($E9C003)

ットが '11' になったベクタ番号が CPU に渡されます。

ジョイスティック

標準で用意されたジョイスティックインタフェースは，いずれもアタリ社の規格に準じたものとなっています。X 68000では，サイバースティックの接続など，汎用のデジタルI/Oとしても利用される傾向にあります。

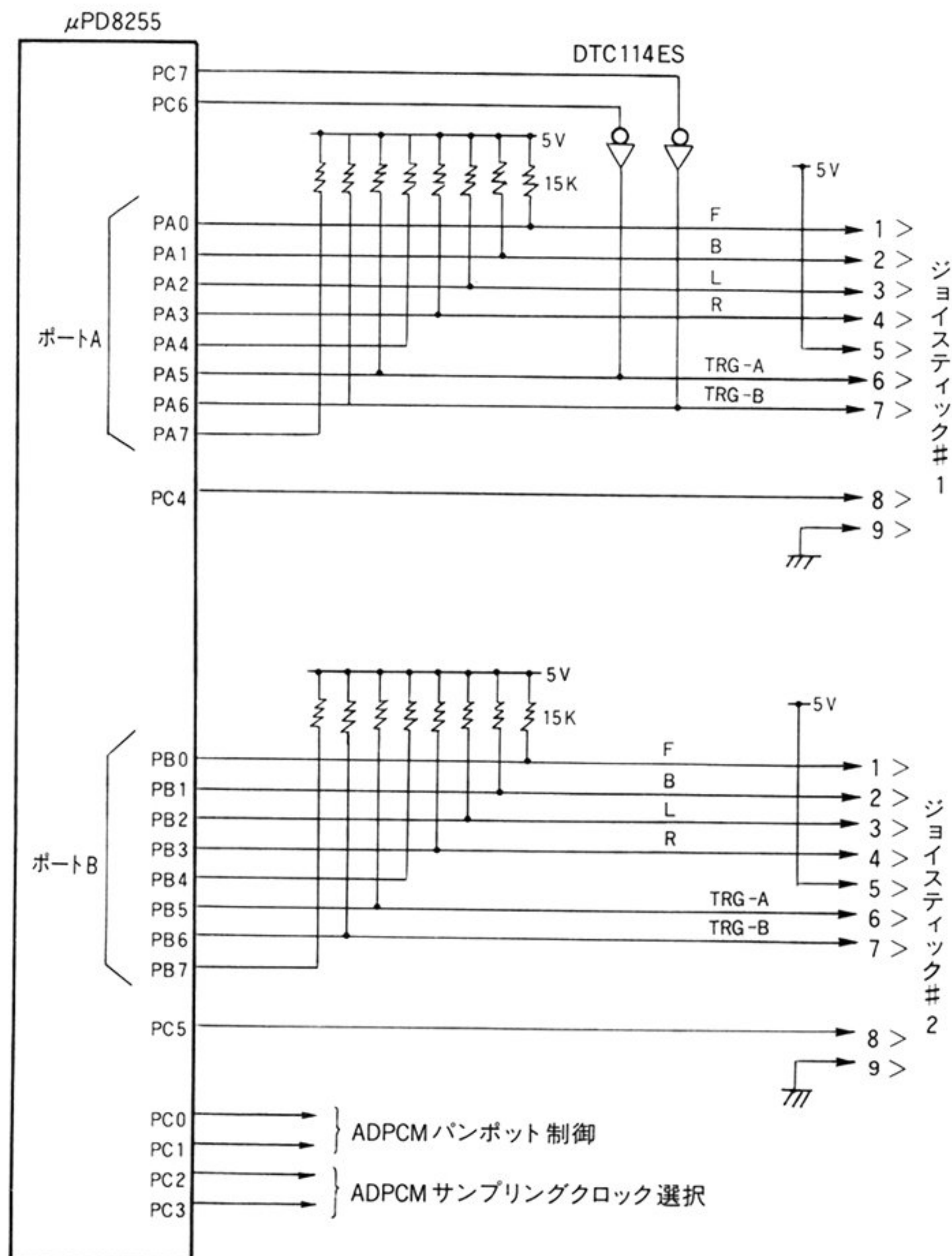
1 ジョイスティックインタフェースの概要

X 68000のジョイスティックインタフェースのブロック図を378ページの図1に示します。ジョイスティックインタフェースは8ビットのパラレルポート（ポートA，ポートB，ポートCと名前がつけられています）を3つ持っているLSI，μPD8255を使用しています。このうちポートCの下位4ビット（PC 0～PC 3）は，ADPCMのパンポット制御やサンプリングクロックの選択に使用し，残りをジョイスティックに割り振っています。

X 68000にはジョイスティックコネクタが2つ設けられていますが，このうちジョイスティック#1はμPD 8255のポートAとPC 4，PC 6，PC 7で，ジョイスティック#2はポートBとPC 5で制御されるようになっています。

ポートA/ポートBは，ジョイスティックのレバーの向いている方向やトリガボタンの状態を読み出すものです。PC 4，PC 5はジョイスティックの操作有効/無効制御に使用され，このビットが'1'（'High'レベル）になっていると，ジョイスティックはスティックやボタンの状態を通知しなくなります。

ジョイスティック#1は，PC 6，PC 7によってオプション機能付きのジョイスティックにも

●図……1　ジョイスティックインタフェースブロック図

対応しているのですが，一般的な4方向＋2トリガタイプのジョイスティックでは，この機能を必要としませんので，どちらのジョイスティックコネクタでも使用できます。

●2 ジョイスティック関連ポート

ジョイスティックに関係するポートの一覧を図2に示します。ジョイスティックはμPD 8255 ひとつだけで制御されており，割り込みの発生機能などもありませんので，すべてμPD 8255 のポートになっています。

●図……2　ジョイスティック関連ポート

ポート	アドレス	bit 7	6	5	4	3	2	1	bit 0	備考
8255ポートA	$E9A001		TRG B	TRG A		RIGHT	LEFT	BACK	FORWARD	ジョイスティック#1
8255ポートB	$E9A003		TRG B	TRG A		RIGHT	LEFT	BACK	FORWARD	ジョイスティック#2
8255ポートC	$E9A005	IOC7	IOC6	IOC5	IOC4	Sampling RATE		PCM	PAN	ジョイスティックコントロール
8255コントロールワード	$E9A007									8255動作モード/ビット操作

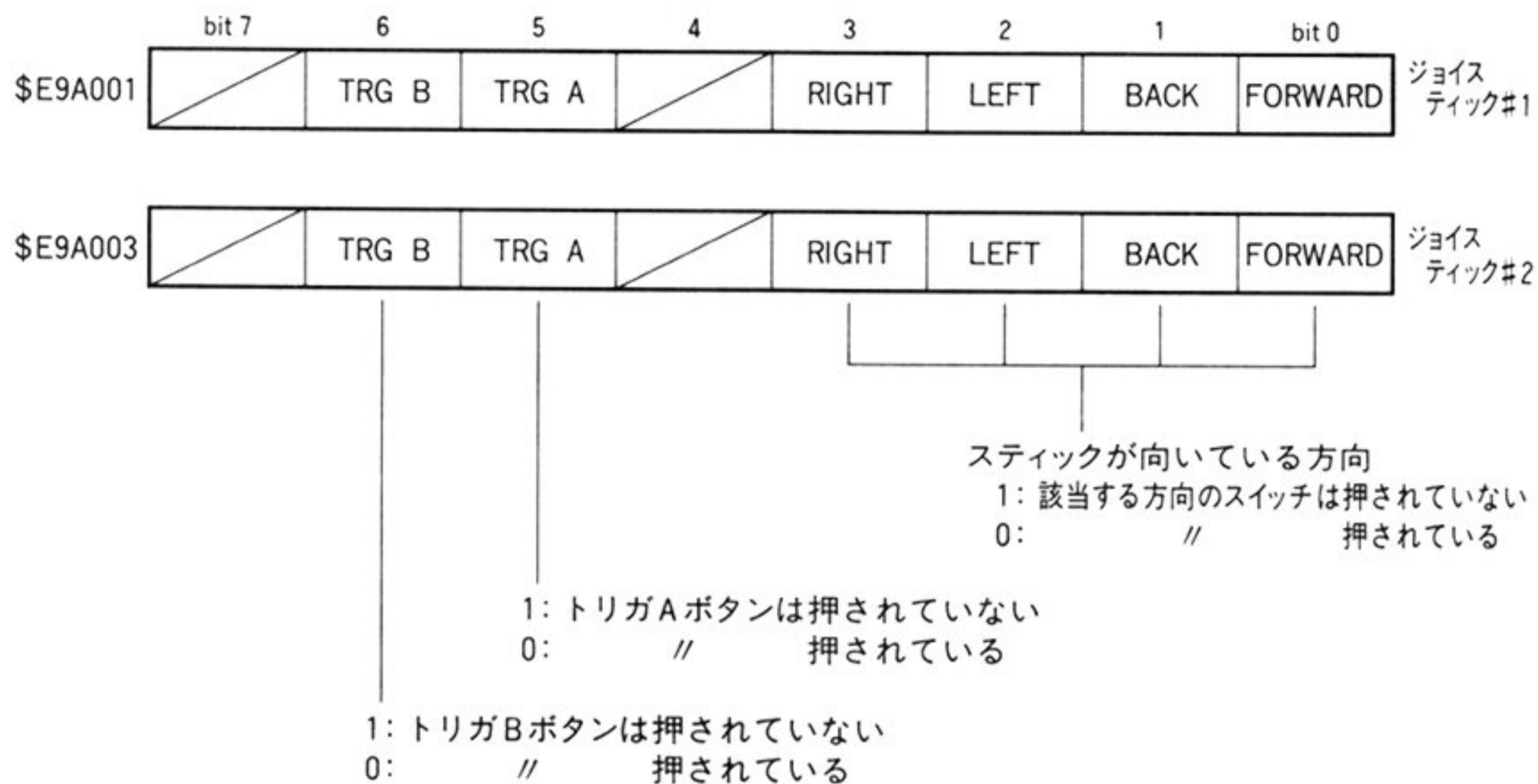
❷·1 ジョイスティック#1/#2

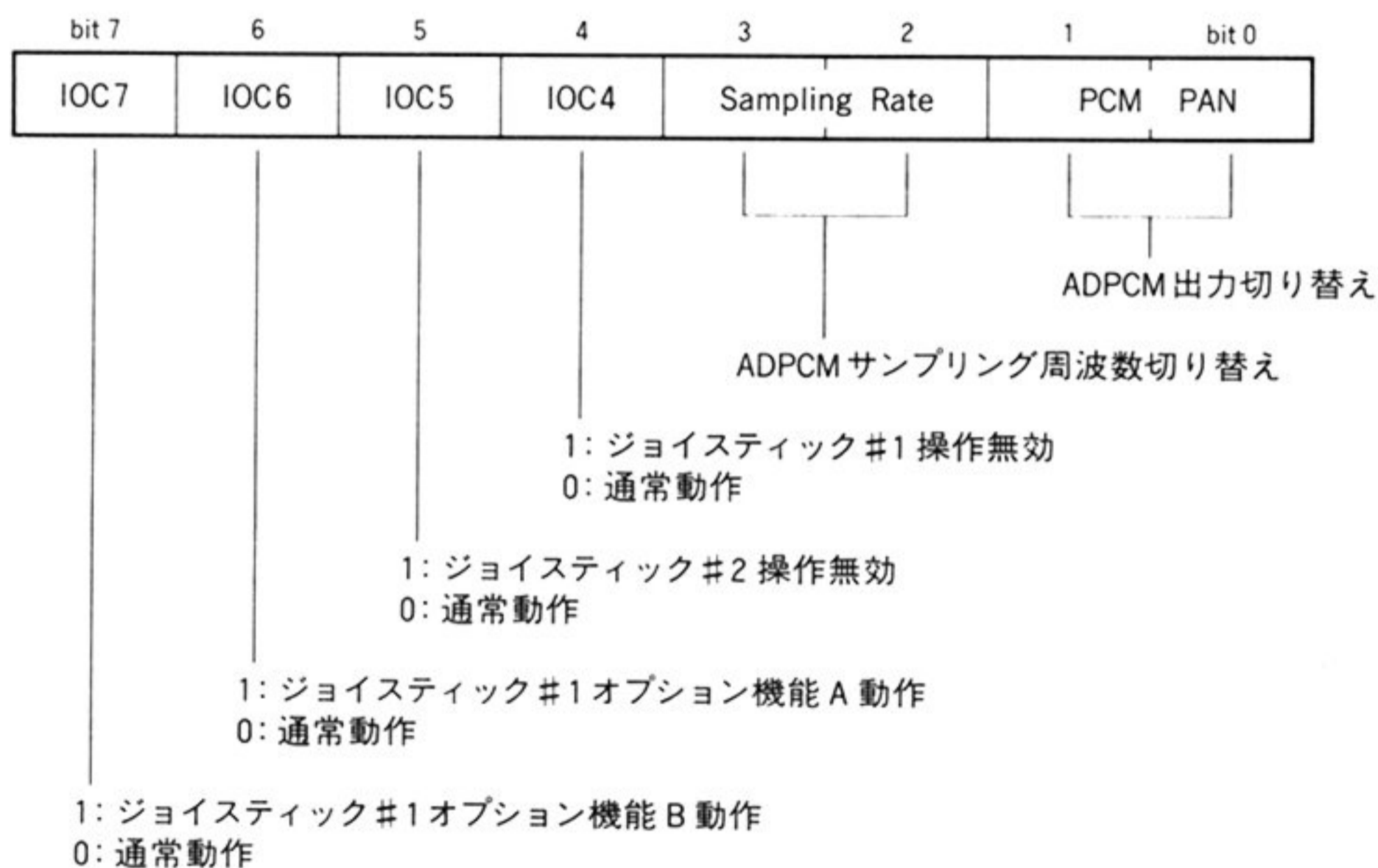
ポートA，ポートBはジョイスティックの状態を読み出すポートです。ビット配置は380ページの図3のようになっています。下位4ビットがスティックの方向を示すデータで，スティックが傾けられると，その方向に取り付けられたスイッチがONになり，'0'が読み出されます。また，ビット5とビット6は，それぞれトリガボタンのAボタン，Bボタンに対応しており，ボタンが押されると，'0'が読み出されます。

ビット4とビット7はブロック図でも示したとおり，コネクタには出力されておらず，抵抗でプルアップされているだけなので，つねに'1'が読み出されます。

❷·2 ジョイスティックコントロール

ポートCは，ジョイスティックの操作の有効/無効制御やADPCMのパンポット制御などに使用されています。ビット配置は図4のようになっています。

ジョイスティックの制御に関係するのは上位4ビットで，このうちビット4，5は，'1'を書

●図……3　ジョイスティック#1/#2

●図……4　ジョイスティックコントロール($E9A005)

き込むとジョイスティックの操作状態が入力されなくなります。

ビット6，7は，ジョイスティック#1が持っているオプション機能用のビットで，トリガボタン信号を出力として利用するものです。このビットに '1' を書き込むと，トリガボタン用の信号線が '0'（Low レベル）になります。通常，このビットは '0' にするようにしてください。

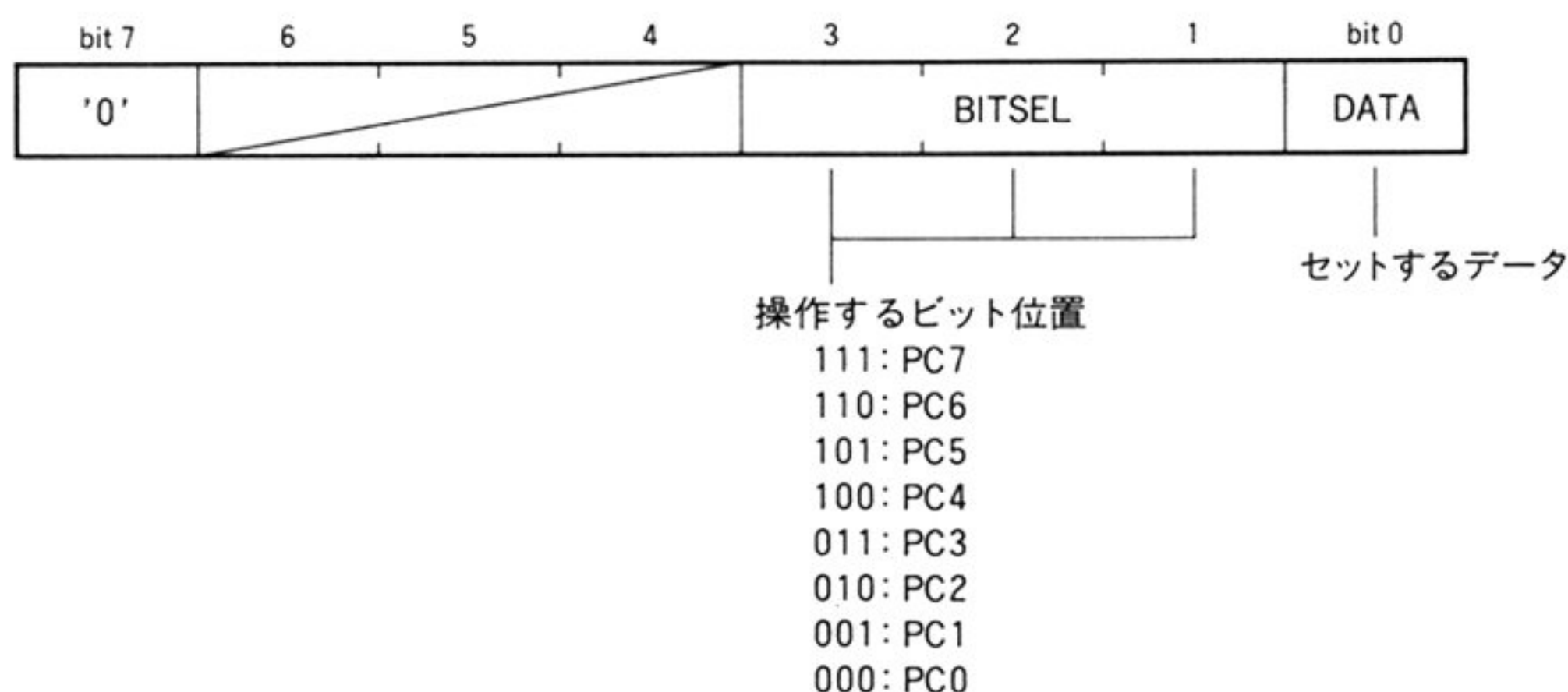
❷・3 コントロールワード

コントロールワードレジスタは，μPD 8255の初期設定やビットセット/リセット機能の制御に使用します。μPD 8255はたんなる入出力ポートとしての動作のほか，パラレルデータ伝送に対応したようなモードも持っています。X 68000では，μPD 8255はジョイスティックインタフェースとなっていることや，下位4ビットがADPCMに使用されているため，データ伝送に使うのは少々苦しいようですが，一応それらについても説明しておくことにします。

❷・❸1 ビットセット/リセットモード

コントロールワードに書き込まれるデータの最上位ビットが'0'になっていると，μPD 8255はビットセット/リセットコマンドとして受け取ります。ビットセット/リセットコマンドでは，ポートCのうち，出力として動作している任意のビットを'1'や'0'に設定できるものです。このときのコマンドフォーマットを図5に示します。

ビット1～3でPC 0～PC 7のいずれを操作するのかを，ビット0でそのビットに設定する値を指定します。

●図……5　コントロールワード（ビットセット/リセット）($E9A007)

❷・❸ 2 モード設定コマンド

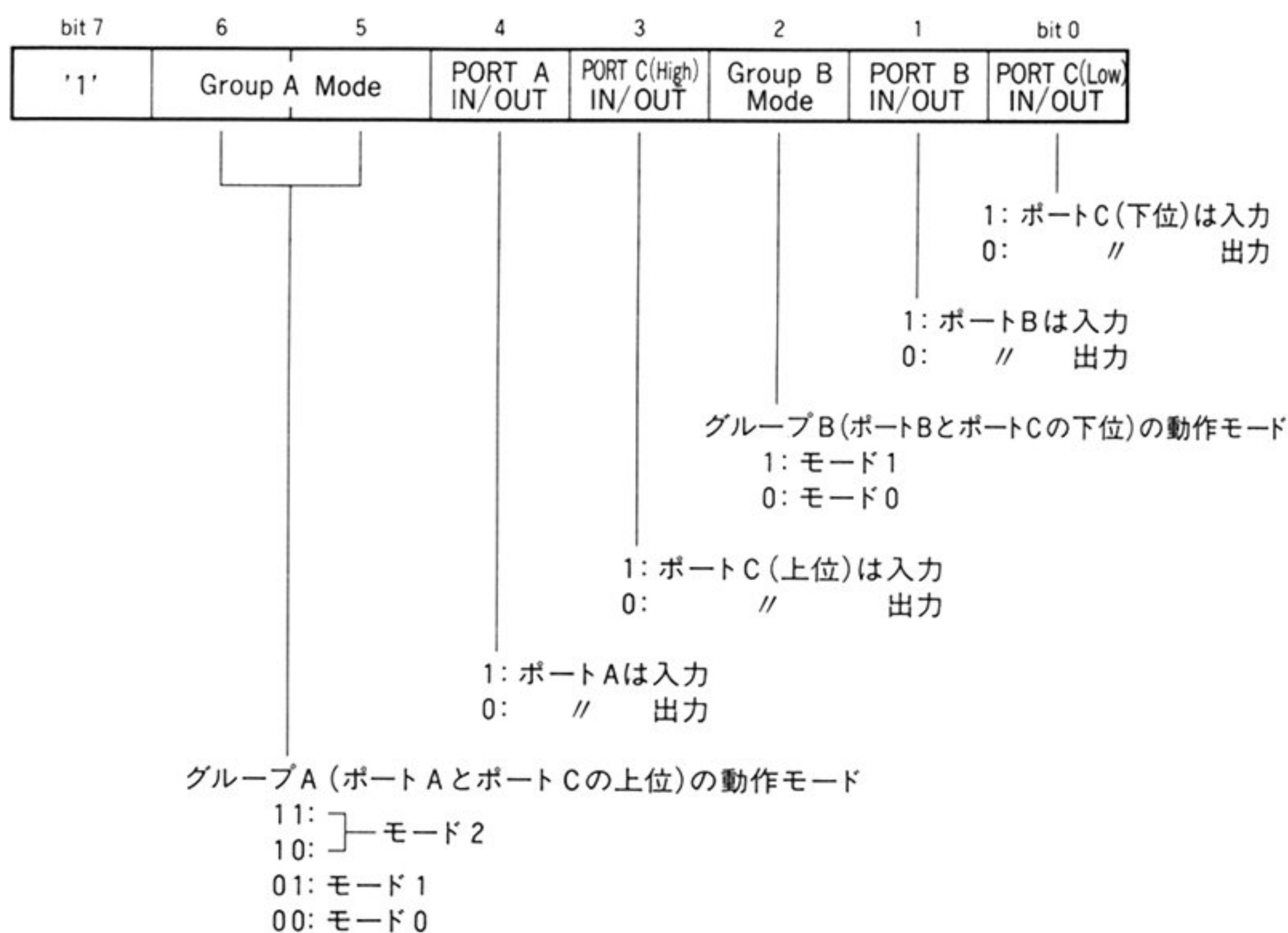
コントロールワードに書き込まれるデータの最上位ビットが'1'になっていると，μPD 8255の動作モード設定コマンドになります。このフォーマットを図6に示します。

μPD 8255は，3つ持っているポートを，大きく2つのグループに分けています。ポートAとポートCの上位がグループA，ポートBとポートCの下位をグループBと呼んでいます。このうちグループAは，動作モードを3つの中から1つ，グループBは2つの中から1つを選択することができるようになっています。

コントロールワードでは，ビット0～2がグループB，ビット3～6がグループAのモード設定になります。

1 モード0

モード0はもっとも単純な入出力ポートとしてプログラムするものです。X 68000では，通常グループA，グループBともこのモードで使用します。このモードではポートA，ポートB,

●図……6　コントロールワード（モード設定）($E9A007)

ポートCの上位4ビット，ポートCの下位4ビットのそれぞれについて入力にするか，出力にするかを個別に設定できます。

X 68000では通常，ポートAとポートBはともに入力，ポートCは上位，下位とも出力として使用しますので，コントロールワードには$92を書き込みます。

X 68000のインタフェースでは，ポートA，ポートB，PC 4，PC 5はたんにコネクタと直結されているだけなので，ジョイスティックポートに自作の周辺装置などをつなぐときに，ポートAやポートBを出力としたり，ポートCの上位（PC 4とPC 5）を入力として使用することも可能です。

2 モード1

モード1は，プリンタインタフェースのような，パラレル伝送をサポートするモードです。制御信号としてポートCを使いますので，X 68000では，このモードが使用できそうなのはグループA側だけです。ですから，ここではグループAについて説明しておきます。

モード1では，入力用として動作するか出力用として動作するかは，コントロールワードのビット4（ポートAのIN/OUT）で決まります。

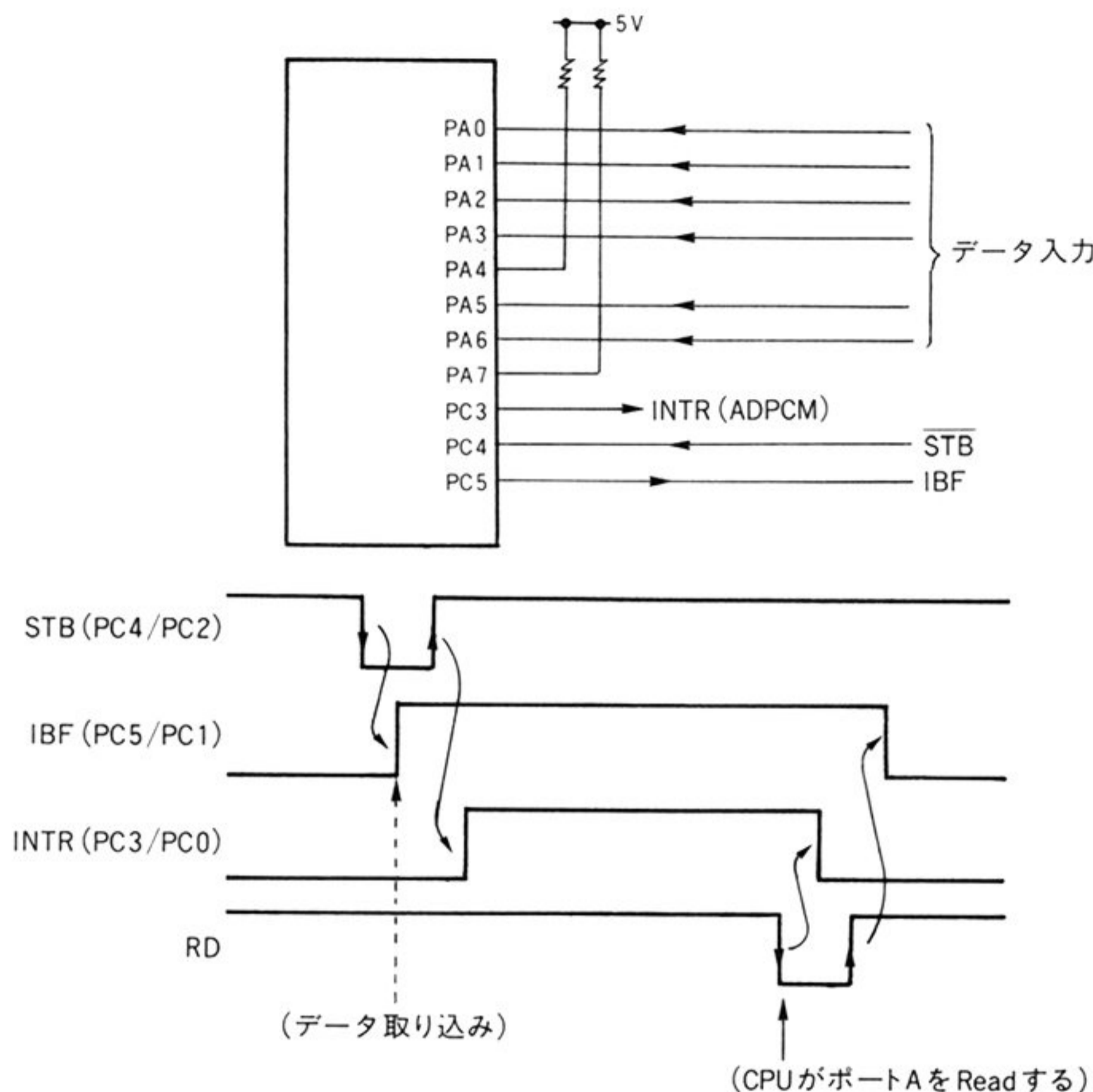
入力用としてプログラムしたときと，出力用としてプログラムしたときの動作を384ページの図7と図8に示します。図では制御信号類は，一応グループA，グループBの両方のものを記入しておきましたので，自作機器でμPD 8255を使うようなときの参考にしてください。

入力動作時

入力用にしたときは，PC 4が外部からのデータ引き取り要求信号（STB），PC 5がバッファにデータが入っているかどうかを示す信号（IBF：Input Buffer Full）として，PC 3がCPUへの割り込み信号（INTR）として動作します。X 68000ではPC 3はADPCMにとられているため，割り込み発生は行えませんが，ステータスチェックでなんとか動作させることはできるでしょう。

外部からデータをPA 0～PA 7にセットしてSTBを'Low'にすると，μPD 8255はデータを取り込むのと同時にIBFを'High'にします。相手がこれを見てSTBを'High'に戻すと，μPD 8255はINTRを'High'にします。CPUがポートAを読み出すと，INTR，IBFとも自動的に'Low'レベルに復帰しますので，相手はデータがCPUに引き取られたことがわかります。

このタイミングをプリンタの動作タイミングと比べると，非常によく似ていることがわかるでしょう。

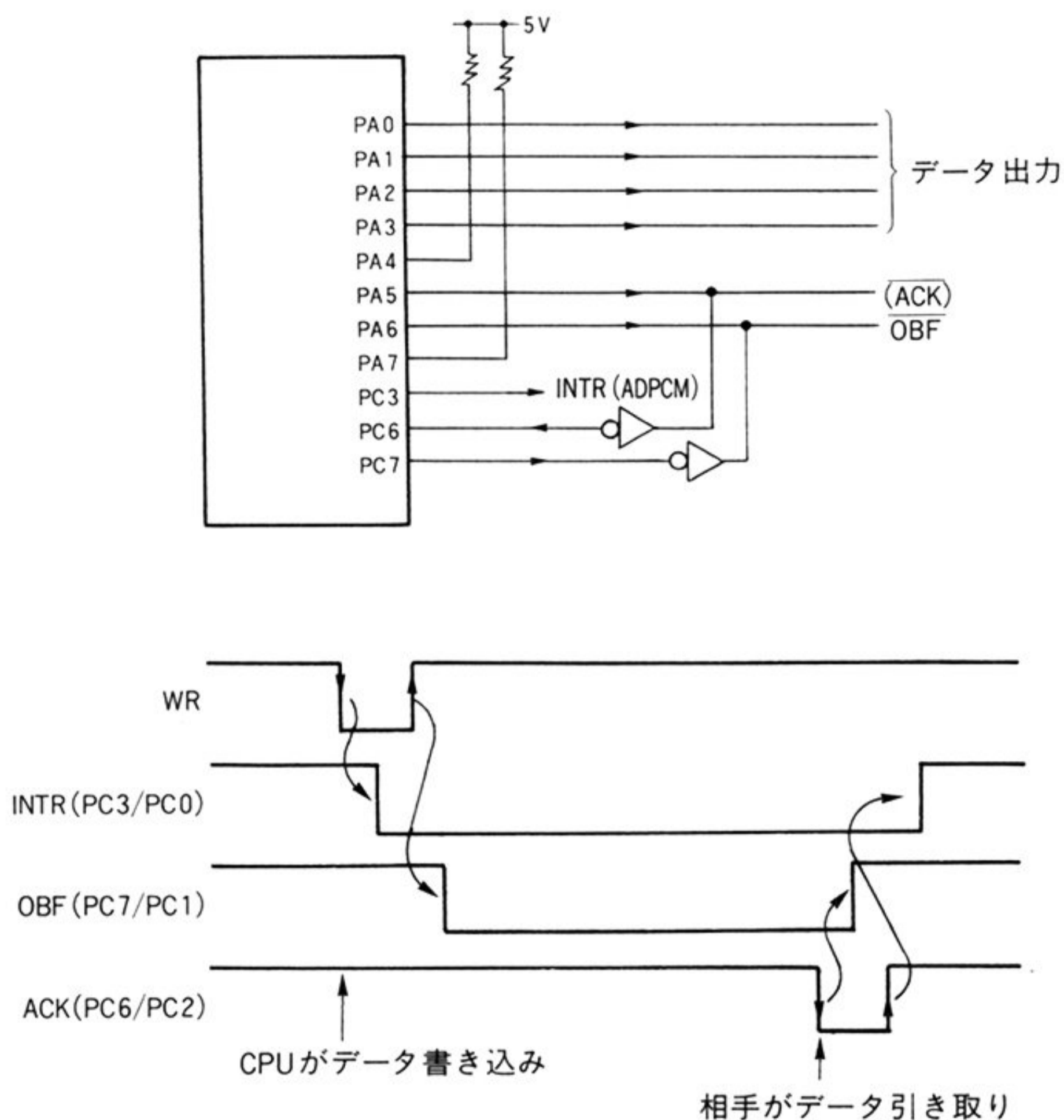
●図……7　モード1（入力モード）

出力動作時

このモードでは，PC 3がCPUへの割り込み要求信号(INTR)，PC 7が出力データセット完了ステータス信号（OBF：Output Buffer Full)，PC 6が相手からデータ引き取り完了信号（ACK）になります。出力動作にプログラムした場合にはPC 6が相手からの入力信号になるのですが，X 68000ではPC 6は出力としてしか使用できませんので，このモードは使えません。

データをポートAに書き込むと，INTRが'Low'になるとともに，OBFが'Low'になり，出力データが用意されたことを示します。相手がこれを受け取り，ACKを返すと，OBFは'High'に復帰し，さらにINTRも'High'になり，CPUへの次のデータセット要求割り込みとなります。

3 モード2

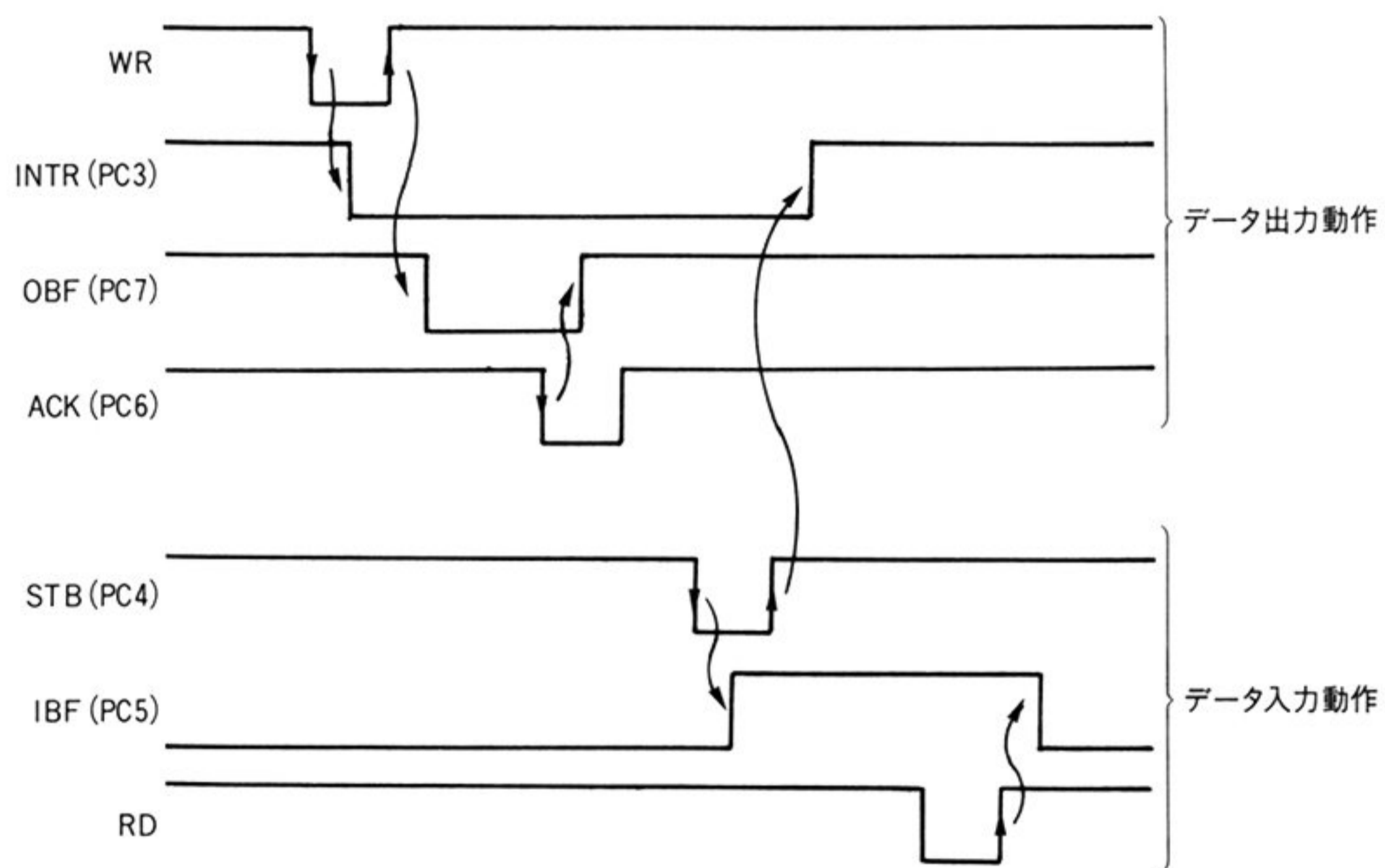
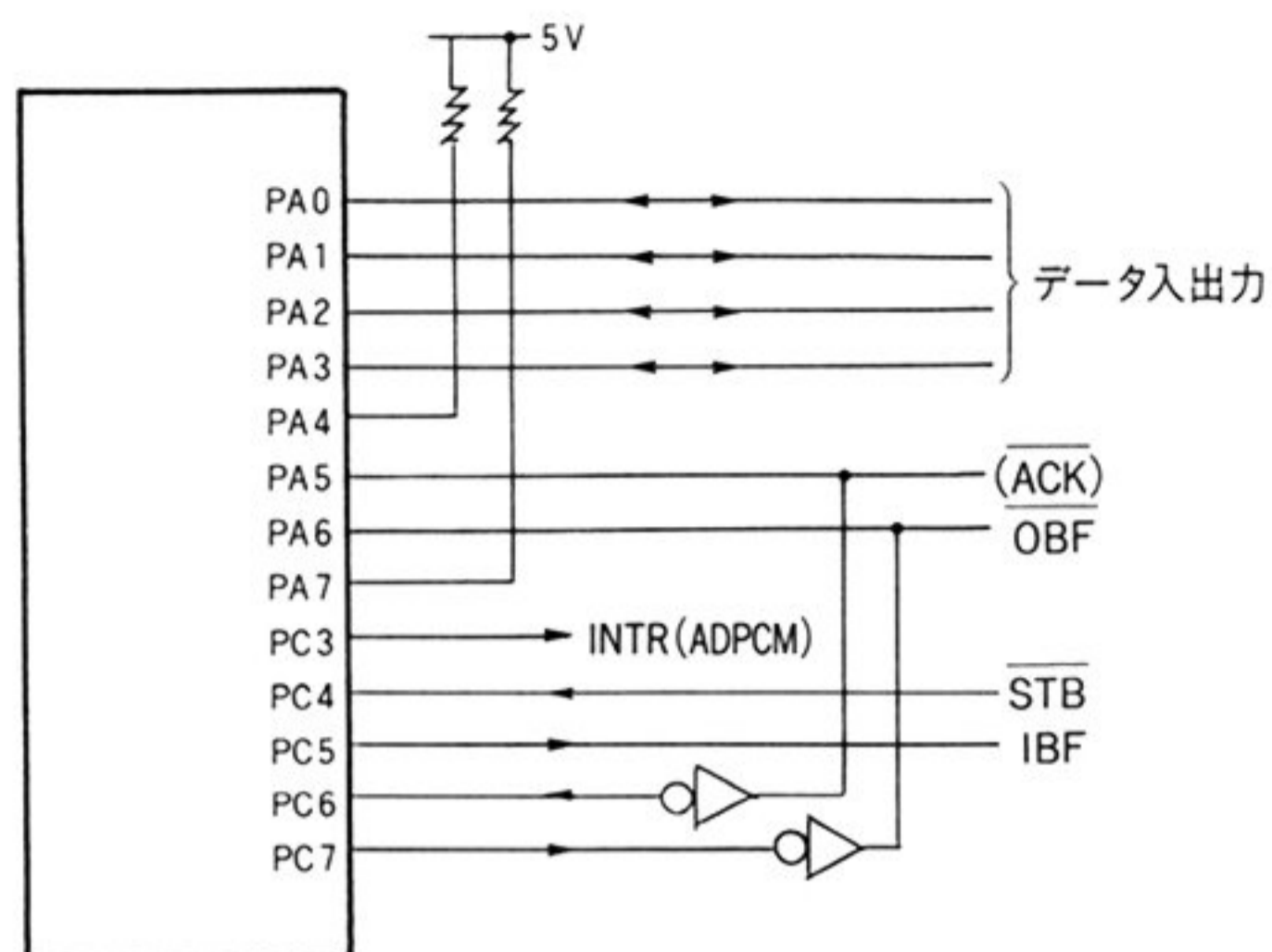
このモードはグループAでのみ使用可能です。このモードは入出力双方向動作が可能です。

●図……8　モード1（出力モード）

＊X68000では，PC6は出力専用のため，このモードは使用不可

このモードでの動作を386ページの図9に示します。ちょうど，モード1の入力モードと出力モードが合体したような動作モードです。

出力動作では，ポートAへの書き込みによってINTRとOBFがともに'Low'となり，相手からのACKがくると，OBFが'High'に復帰します。

逆に相手からSTBを受けると，INTRとともにIBFが'High'になり，データが取り込まれたことを示し，CPUがポートAを読み取ると，IBFが'Low'に復帰します。

●図……9　モード2

＊X68000では，PC6が出力専用のため，データ出力動作は不可

フロッピーディスクドライブ

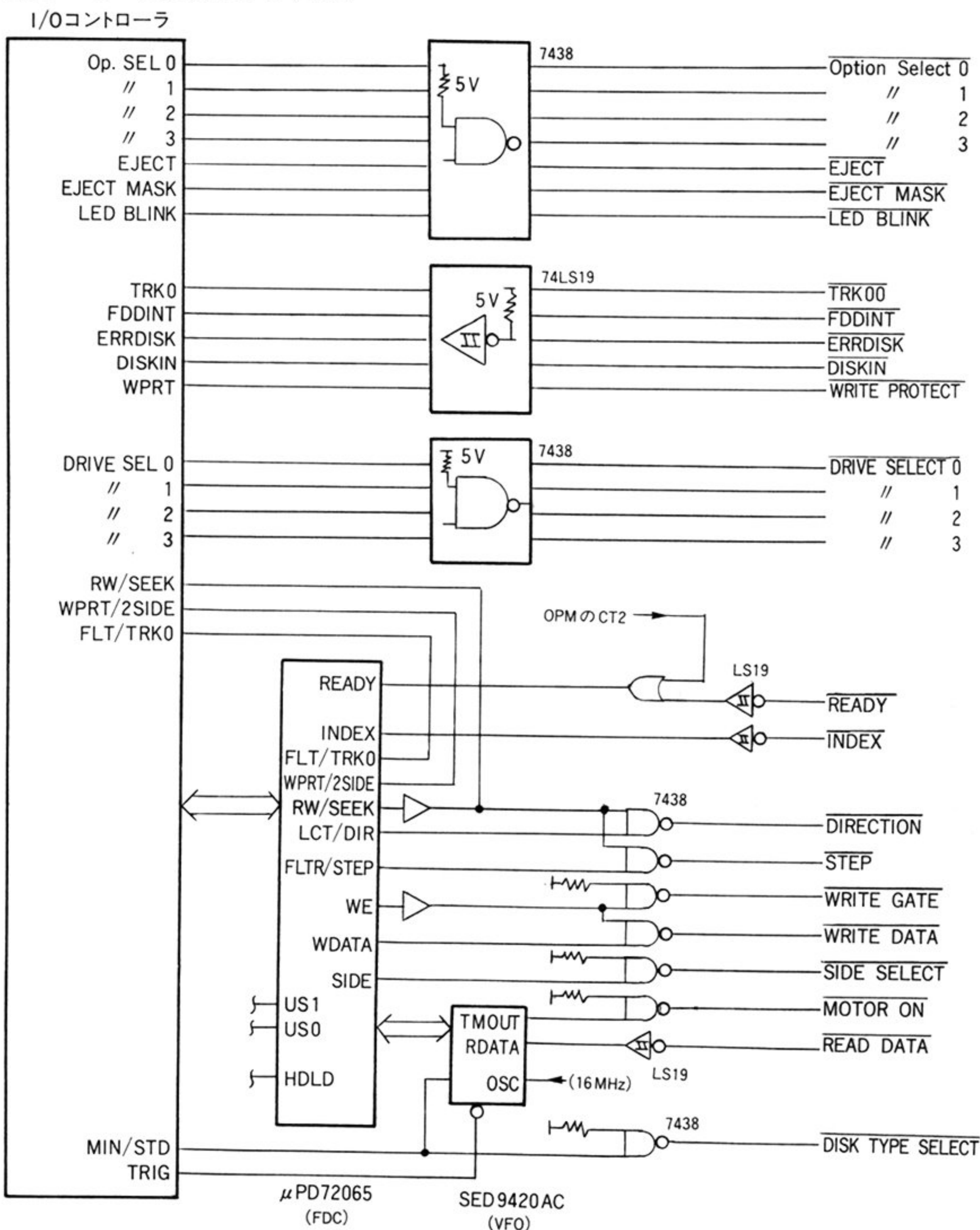
レバーを廃止し，オートイジェクト機構を取り入れたFDDを，X 68000ではじめて目にしたという方も多いのではないでしょうか。ここでは，ディスクのリード/ライトのほか，FDDの持つ機能について説明します。

1　FDDインタフェースの概要

X 68000のフロッピーディスクドライブインタフェースのブロック図を388ページの図1に示します。ディスクとのリード/ライト制御を行うLSI(FDC：フロッピーディスクコントローラ)には日本電気製のμPD 72065を使用しています。その横にあるSED9420AC (VFO)は，ディスクから読み出された波形からデータとクロックを分離するICで，データセパレータとも呼ばれます。

一般的なFDD（フロッピーディスクドライブ）であれば，インタフェースはFDCとVFOだけで十分なのですが，X 68000が使用しているドライブにはディスクを挿入すると自動的にディスクをクランプするオートクランプや，ソフトウェアでディスクを排出するオートイジェクト機能などが追加されているため，これらの制御やFDDの状態変化検出などをI/Oコントローラ（シャープがX 68000用につくったLSI）でサポートするようにしています。

また，プログラムを作成するうえで注意すべき点としては，ドライブセレクト信号がI/Oコントローラから出力されていることと，OPM（FM音源IC）のCT 2出力を使ってFDCのREADY端子を強制的に'1'にする機能が追加されている点があげられます。

●図……1　FDD周辺ブロック図

μPD 72065は，US0，US1という信号を使って4台までのFDDにディスク選択信号を出力することができ，コマンド中にドライブ番号を指定すれば，この信号を使ってドライブ選択を行ってくれるのですが，X 68000ではこの信号は使用せず，I/Oコントローラから出力するようにしています。ディスクアクセスのときにはFDCにコマンドを書き込む前にI/Oコントロ

ーラのレジスタでアクセスするドライブを選択しておく必要があります。

FDCのREADY端子は通常はFDD側のREADY信号と直結されており，FDDがアクセス可能な状態（ディスクがクランプされ，モータの回転が安定する）になると'1'になる入力信号ピンです。これをOPMのCT2で強制的に'1'にする機能は，FDDが接続されているか否かをチェックするときに使用するために設けられたもので，通常のアクセスで使用することはありません。

●2 FDDの仕様

X68000の本体に内蔵されているFDDの仕様の概略を390ページの図2に示します。X68000のFDDインタフェースは2HDや2DD/2D，およびそれぞれの単密度フォーマット（8インチや5インチの初期のころに使用されていました）もサポートしているのですが，内蔵FDDがサポートできるのは2HDおよびその単密度フォーマットだけで，2DDや2Dは扱えません。

また，X68000のFDDは，ディスクをクランプするとヘッドがディスクについたままとなりますので，ヘッドのロード/アンロード(ヘッドをディスクに押しつけたり，離したりすること)を考える必要はありません。図1でも示したように，FDCのヘッド制御信号（HDLD）もオープンのままになっています。

●3 FDDインタフェース関連ポート

FDDインタフェースに関連するポートの一覧を390ページの図3に示します。FDDインタフェースはFDCのほか，I/OコントローラとOPM（CT2端子）を使用してつくられています。FDCはディスクのリード/ライト，ヘッド移動などのFDDの基本動作，I/Oコントローラは割り込みやオートイジェクト，LEDなどのオプション機能の制御，OPMのCT2端子はディスクの接続状態検出のために使用されます。

●図……2　本体内蔵FDDの仕様

項目		値	備考
記憶容量	アンフォーマット時	1667KB	
	フォーマット時	1065KB	IBM準拠，高密度モード，256バイト/セクタ，26セクタ/トラック
	トラックあたり容量	10.42KB	
データ転送速度		500Kbit/s	
アクセスタイム	トラック間移動時間	3ms	シーク時の待ち時間 ＝トラック間移動時間＋シークセトリング時間 平均アクセス時間 ＝平均トラック移動時間＋シークセトリング時間
	シークセトリング時間	15ms	
	平均アクセス時間	95ms	
メディア回転送		360rpm	
スピンドルモータ起動時間		0.5s	
トラック数	TRACK/SIDE	77	
	TRACK/DRIVE	154	
トラック密度		96 TPI	TPI(トラック/インチ)
ヘッド数		2	
変調方式		MFM	FM方式も可
その他		オートクランプ オートイジェクト オートリキャリブレート LED	

●図……3　FDDインタフェース関連ポートアドレス

デバイス	アドレス	READ/WRITE	bit7	6	5	4	3	2	1	bit0	備考
FDC (μPD72065)	$E94001	R									FDCステータスレジスタ
		W									FDCコマンドレジスタ*1
	$E94003	R									FDCデータレジスタ
		W									FDCコマンドレジスタ
I/Oコントローラ	$E94005	R	DISK IN	ERR DISK	'0'						ドライブステータス
		W	LED CTRL	EJECT MASK	EJECT ON/OFF	'0'	DRIVE #3	DRIVE #2	DRIVE #1	DRIVE #0	ドライブオプション信号制御
	$E94007	W	MOT ON	'0'		2HD/2DD	'0'		ACCESS DRIVE		アクセスドライブセレクト等
	$E9C001	R	FDC INT	FDD INT	PRT INT	HDD INT	HDDI EN	FDCI EN	FDDI EN	PRTI EN	割り込み信号ステータス
		W	'0'				HDDI EN	FDCI EN	FDDI EN	PRTI EN	割り込み信号マスク
	$E9C003	W	Vect						DEVICE		割り込みベクタ番号
OPM	$E90003	W	CT1	CT2					W		レジスタ$1B($E90001に$1Bを書き込んでからアクセスする

＊1：SET STANDBY($35)，RESET STANDBY($34)，SOFTWARE RESET($36)以下のコマンドは使用不可

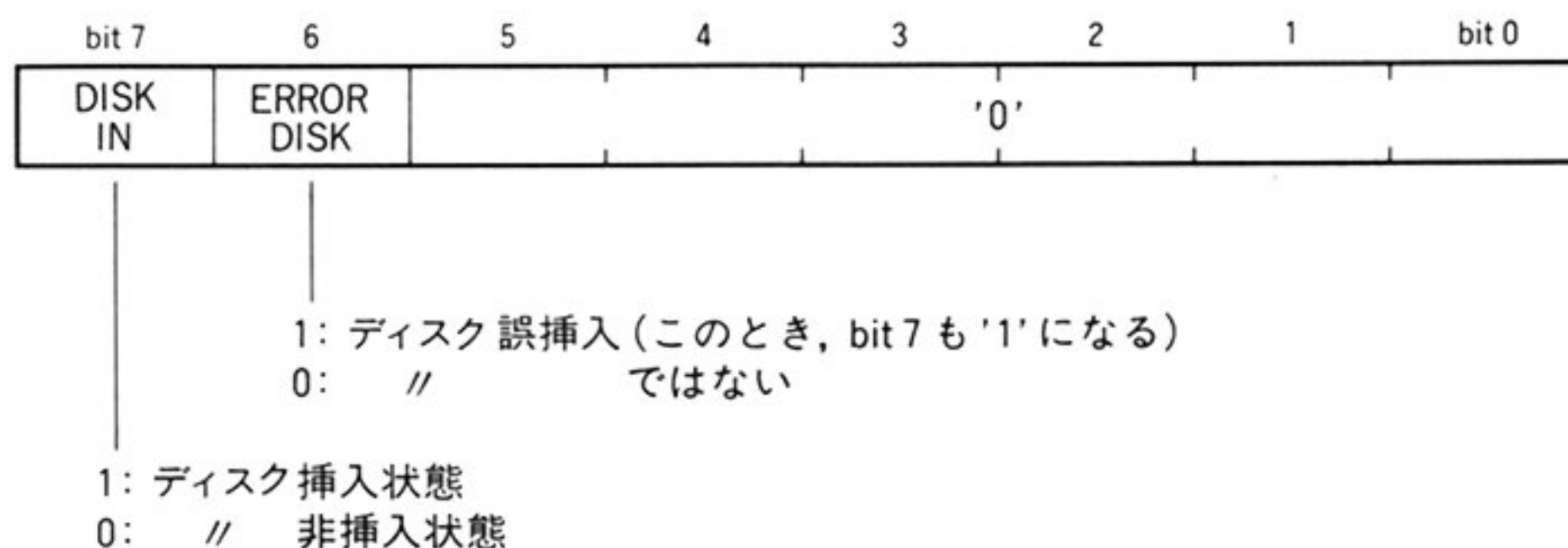
❸·1 I/OコントローラのFDD関連ポート

❸·❶1 ドライブステータスレジスタ

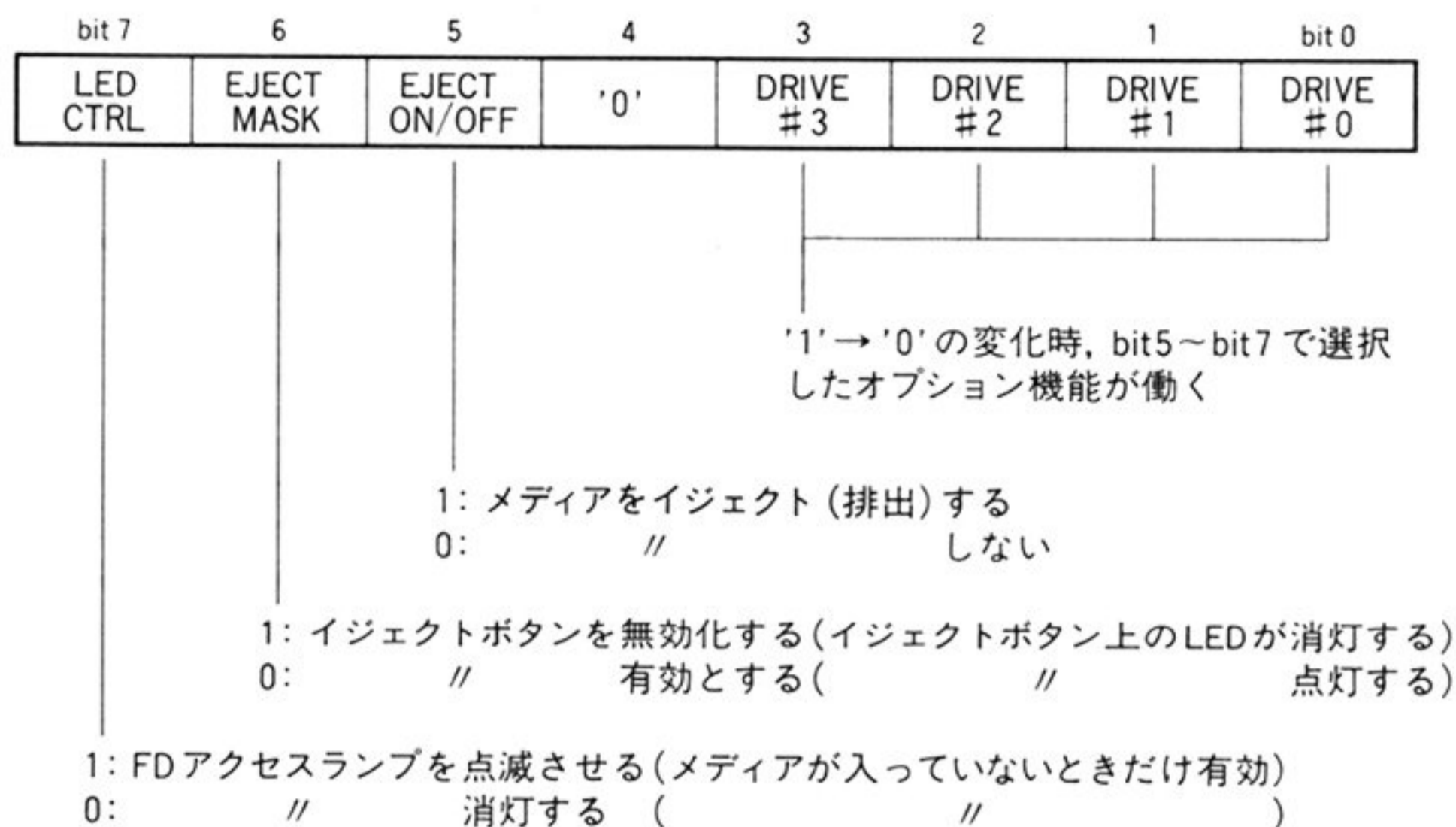
ドライブステータスポートのビット配置を図4に示します。このレジスタはFDDへのディスクの挿入状態を示すものです。ビット7はディスクが挿入されているか否かを，ビット6はディスクの表裏を間違えるなど誤挿入がされているか否かを示すものです。

これらのステータスは，ドライブコントロールレジスタのビット0～3のうち，'1'を書き込んだドライブのものが読み出されます（複数のビットを'1'にすると，'1'にされたドライブのステータスの論理和（OR）をとったものになります)。

X 68000ではディスクの抜き挿しが行われると割り込みが発生します。割り込みが発生したら，各ドライブの状態を順に読み出すことで，どのドライブに変化があったのかを判断することができます。ディスクの誤挿入があった場合には，ディスクの挿入にともなう割り込みが発生した後（ドライブステータスレジスタのビット7は'1'になります)，FDDはCPUの関与を受けず，自動的にディスクを排出します（ビット7は'0'になります)。この排出時にも割り込みが発生しますので，誤挿入があった場合には2回連続して割り込みが発生することになります。

●図……4　ドライブステータス　$E 94005

❸·❶2 ドライブコントロールレジスタ

ドライブコントロールレジスタでは，LEDやイジェクトなどのオプション機能の制御を行

●図……5　ドライブコントロール　$E 94005

います。ビット配置は図5のようになっています。

ビット5～7で各オプション機能を，ビット0～3でオプション機能を働かせるドライブ番号を指定します（内蔵ドライブは0と1です）。各オプション機能は，ドライブ選択ビットが'1'から'0'になったときに動作するようになっています。ドライブの選択は複数を同時に行ってもかまいません。

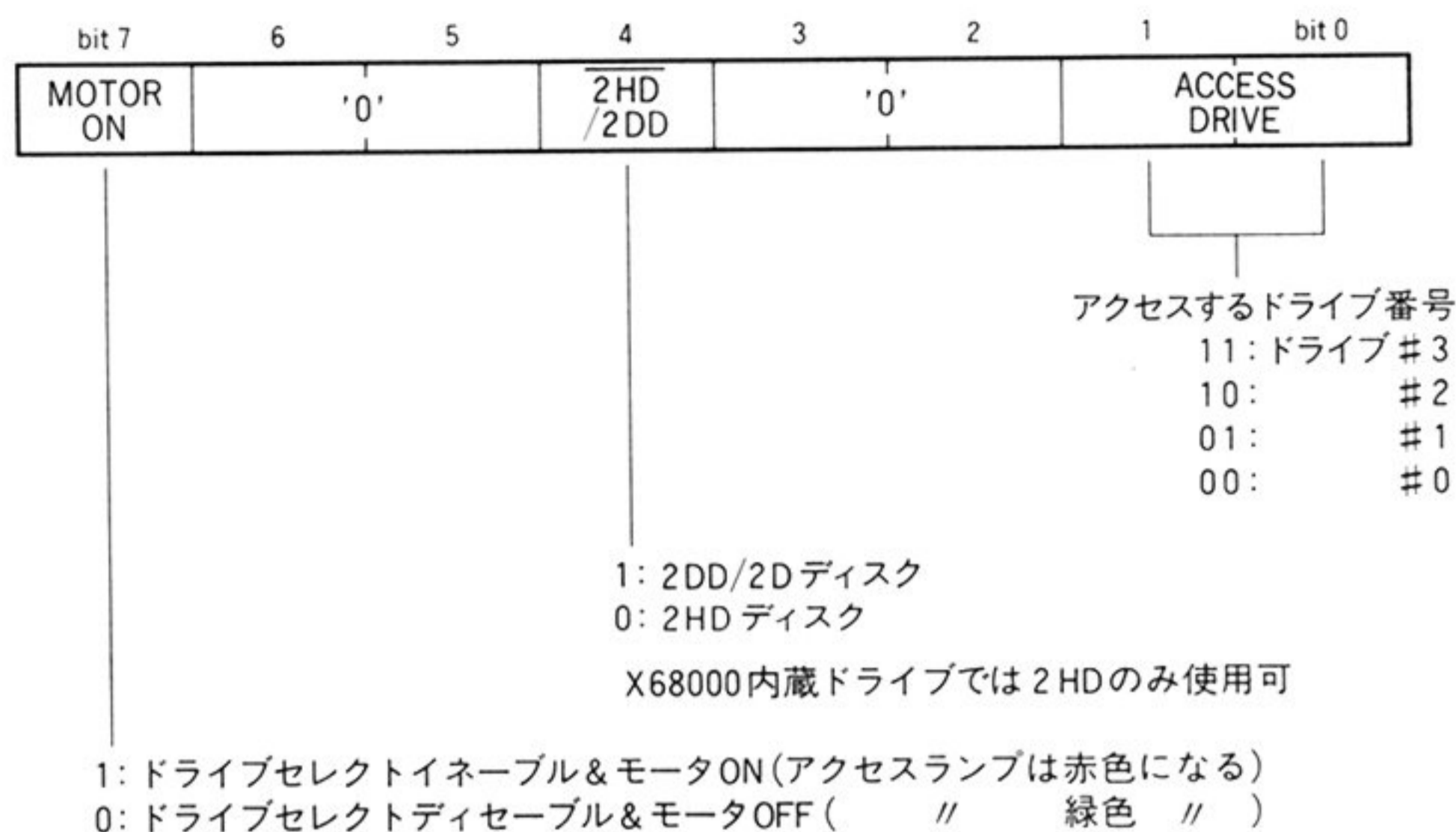
たとえば，このレジスタに$23を書き込んだ後，$20を書き込むと，ドライブ0とドライブ1のディスクが同時にイジェクトされます。

❸・❶ 3　アクセスドライブセレクトレジスタ

レジスタのビット配置を図6に示します。このレジスタはアクセスするFDDのドライブ番号やメディアタイプの選択を行います。

ビット7を'1'にすると，FDDのモータが回転しはじめるとともに，ビット0，1で選択したドライブへのセレクト信号がアクティブになり，アクセスランプ（LED）が緑から赤に変わります。'0'にしてもしばらくは回転したままで，一定時間たってから停止します。アクセスを行う前には必ずビット0，1でドライブ番号を設定するとともに，このビットを'1'に設定してください。

ビット4は2HDと2DDの切り替えを行うビットですが，内蔵ドライブでは2HDしかサポートされないため，このビットは通常'0'のままで使用します。

●図……6　アクセスドライブセレクト　$E94007

❸・❶4 割り込みステータスレジスタ

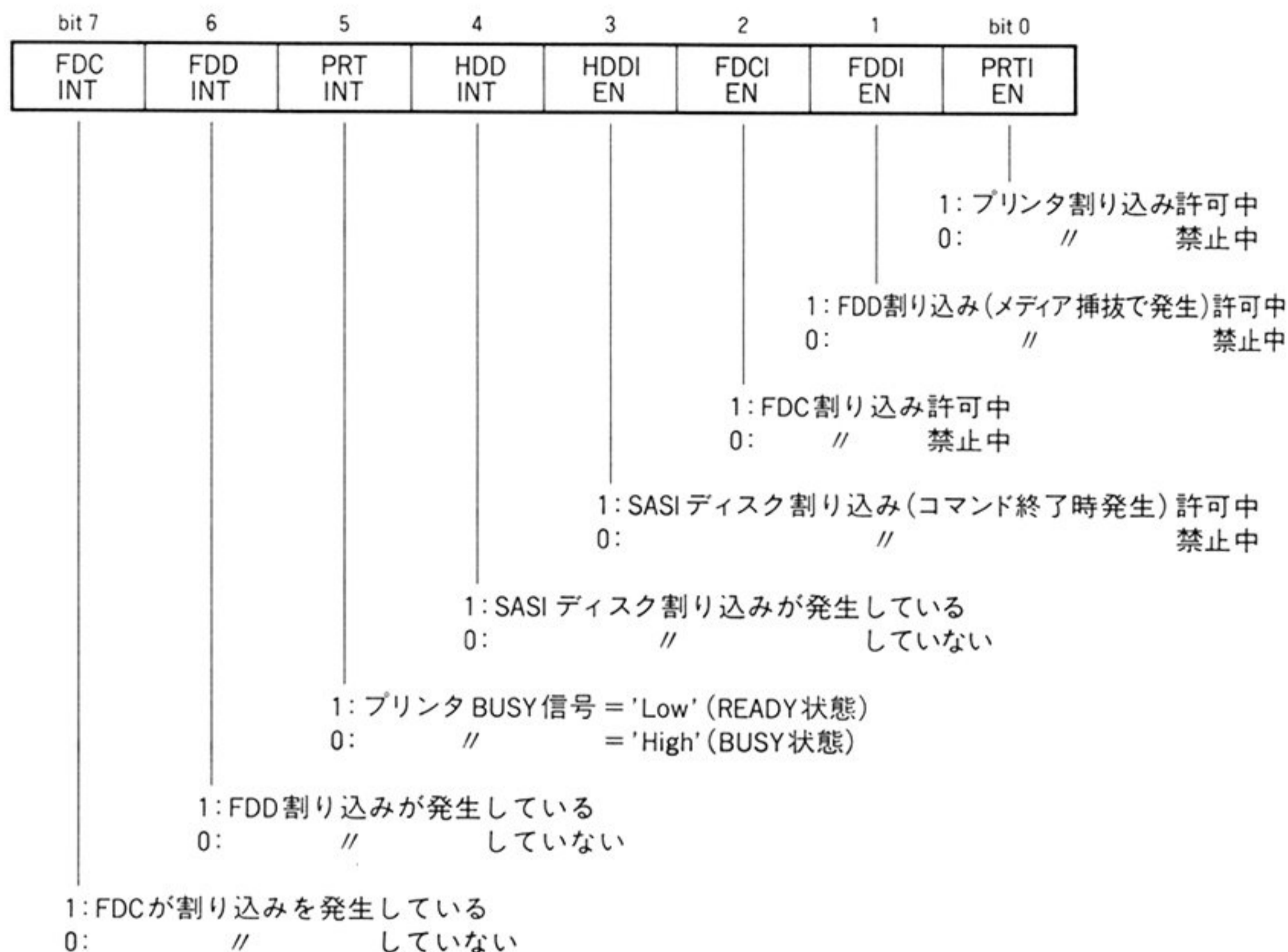
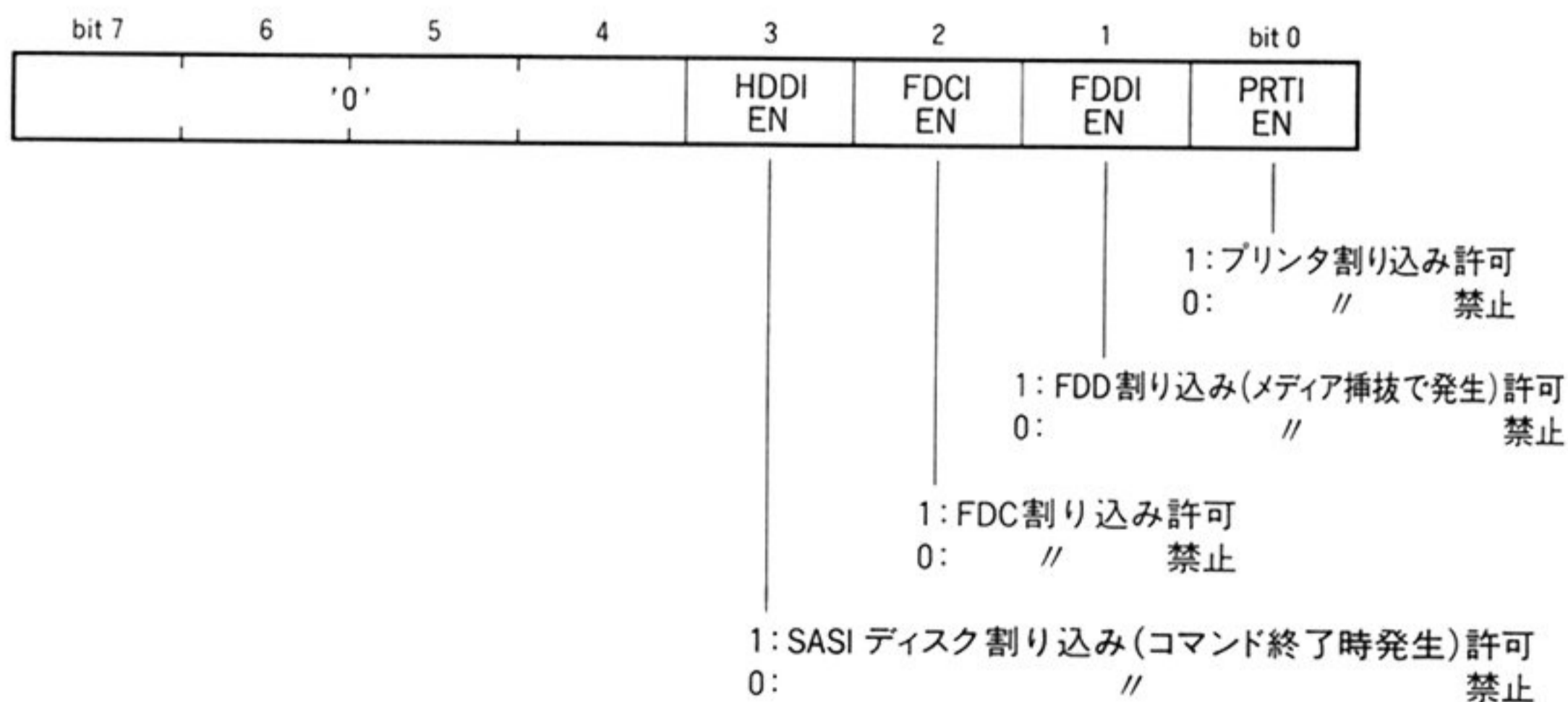
ビット配置を394ページの図7に示します。I/OコントローラはSASI（ハードディスク），フロッピーディスク，プリンタなどのインタフェースを受け持っており，割り込みもI/Oコントローラで管理できるようになっています。I/Oコントローラが管理している割り込みのステータスや許可状態を示すのが，このレジスタです。

上位4ビットは割り込み要求の発生状態を示すもの，下位4ビットは割り込みマスクレジスタの内容がそのまま反映されており，それぞれの割り込みの発生が許可になっているか否かを示しています。

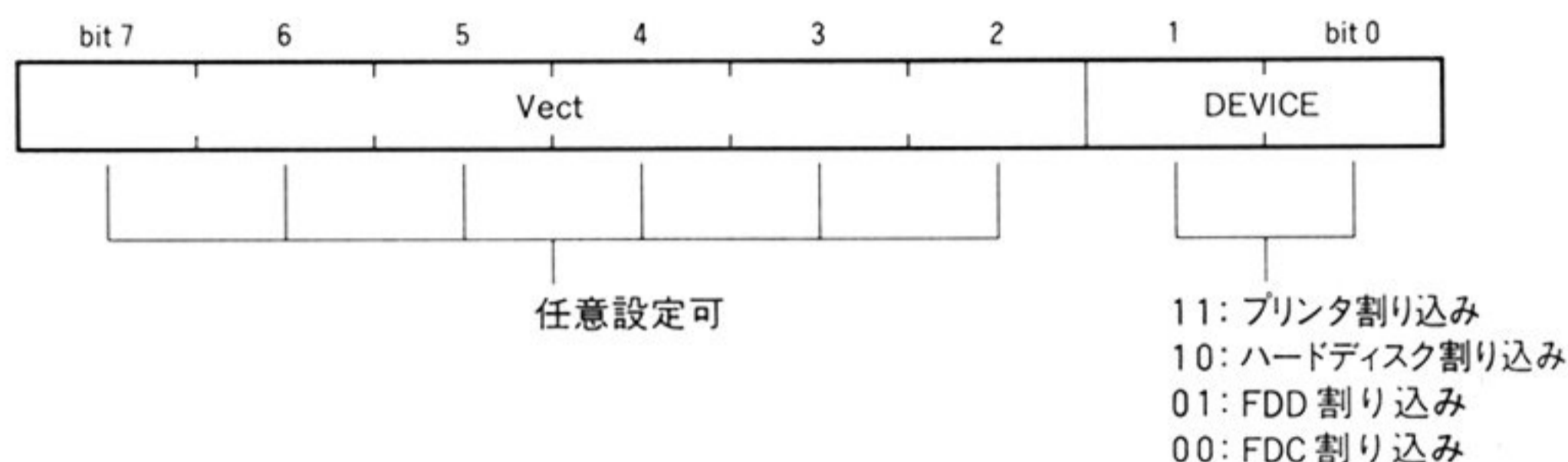
割り込みが許可になっていない('0'になっている)場合，割り込み要求があっても，CPUへの割り込み要求は行いませんが，割り込みの要求状態は下位4ビットで読み出すことができます。

❸・❶5 割り込み信号マスクレジスタ

ビット配置を394ページの図8に示します。I/Oコントローラの管理している割り込みの許可/禁止を制御します。それぞれのビットが'1'のとき割り込み発生が許可，'0'のとき禁止になります。

●図……7　割り込み信号ステータス　$E9C001

●図……8　割り込み信号マスク　$E9C001

❸·❶6 割り込みベクタ設定レジスタ

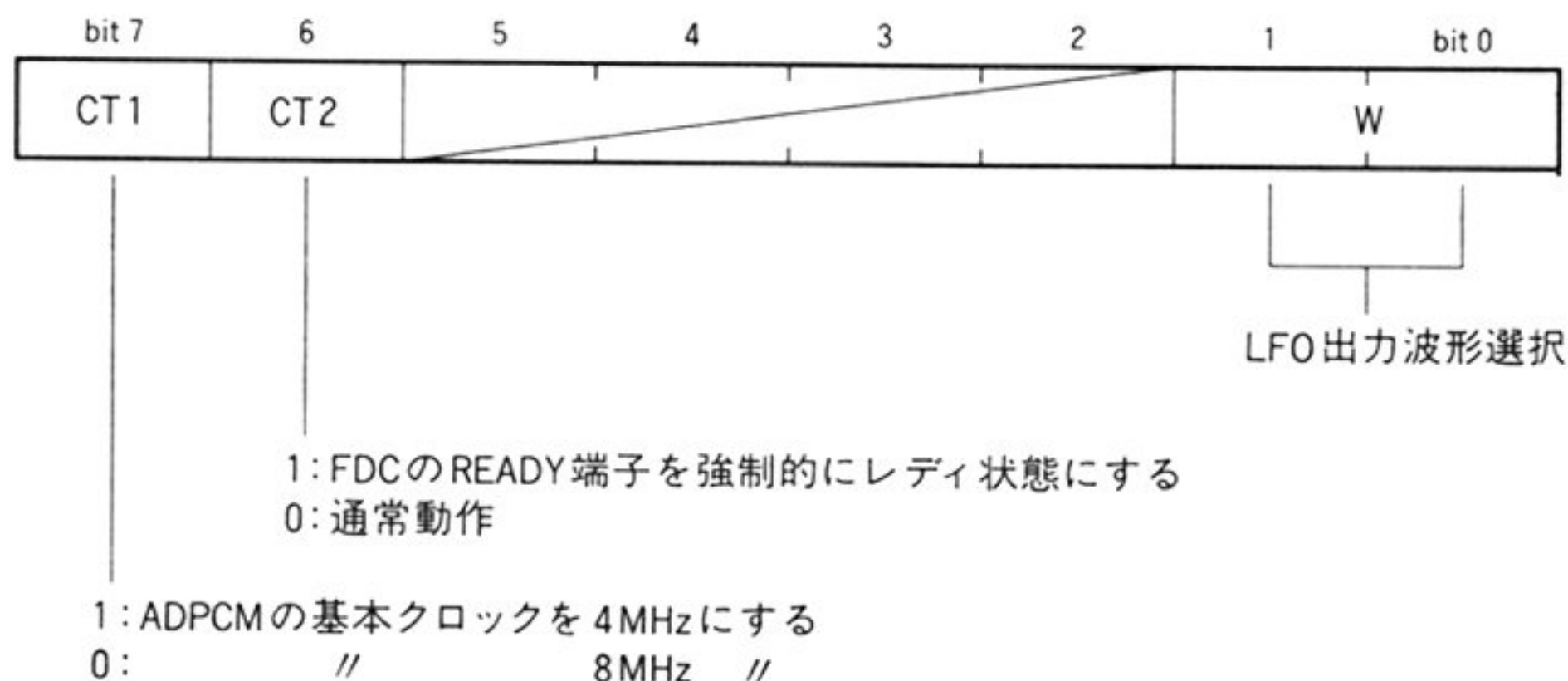
ビット配置は図9のようになっています。このレジスタにはI/Oコントローラが出力する割り込みベクタ番号を設定します。設定が有効なのは上位6ビットで，下位2ビットは割り込み発生時，I/Oコントローラが割り込み要因によって自動的に変更してCPUに与えます。

●図……9　割り込みベクタ設定　$E9C003

❸·2 OPM(YM2151)のFDD関連ポート

OPMのCT 2端子の制御を行うレジスタ$1Bのビット配置を，396ページの図10に示します。ビット6を'1'にすると，FDCのREADY端子が強制的にレディ状態（ディスクがクランプされ，定常回転している状態）になります。

この機能は，ディスクの接続状態のチェックに使用します。このビットを'1'にしてディスクにRECALIBRATEコマンドを発行すると，FDCはドライブがレディ状態にあるものとみなし，ヘッドを0トラックに移動させようとします。ディスクが接続されていれば，FDDからトラック0への移動ステータス信号が検出され，コマンドが正常終了しますが，接続されていないと，いくらヘッド移動パルスを送っても，トラック0が検出できないため，異常終了となるわけです（このビットを'0'にしてRECALIBRATEコマンドを実行すると，ディスクが入っていないドライブはヘッドの移動が行われず，即座にエラー終了します)。

このビットはドライブの接続チェック以外のときはつねに'0'にするようにしてください。

●図……10　OPMのレジスタ　$1B($E90003)

4 FDC

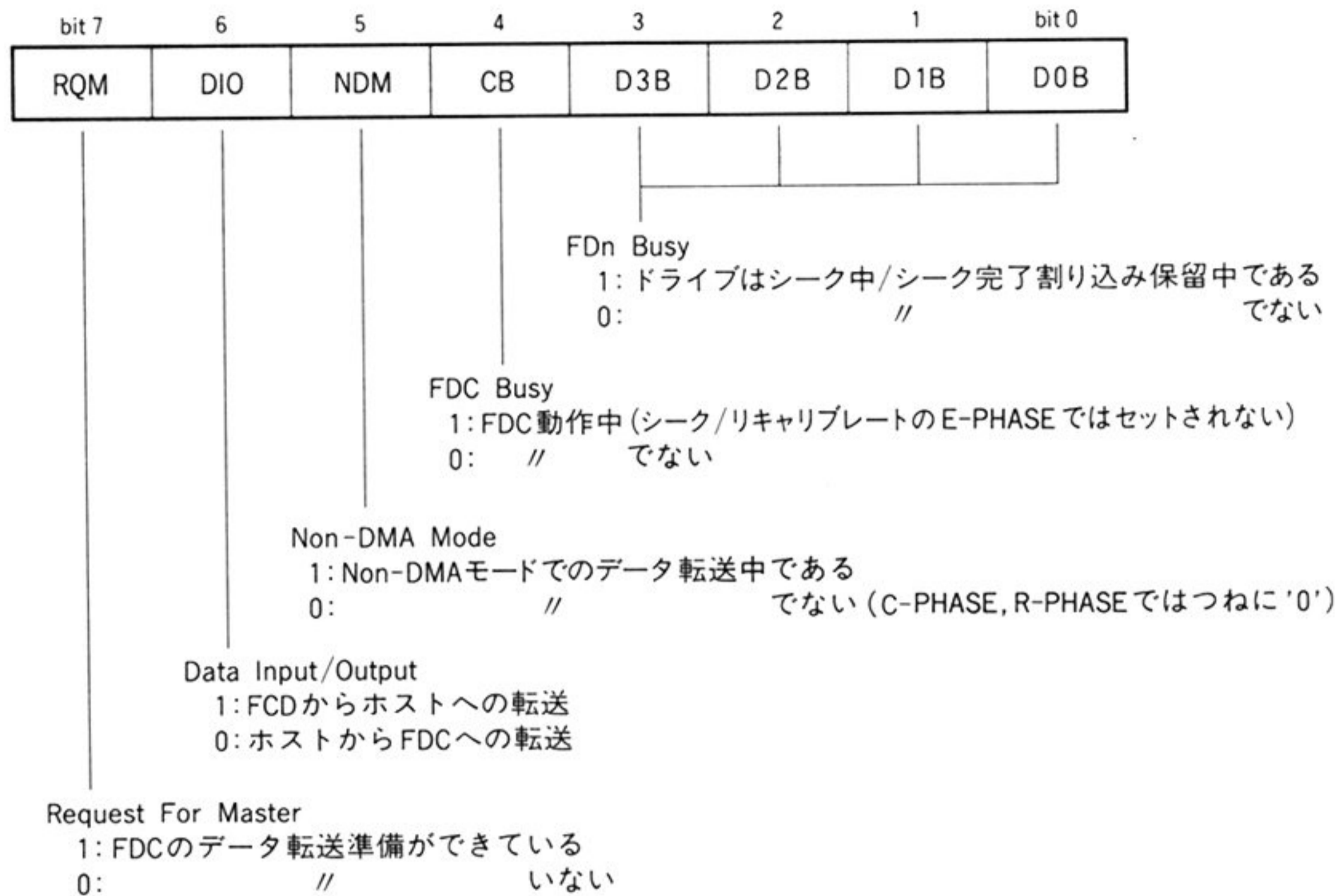
FDCのポートは$E94001と$E94003番地に割り振られています。FDCへのアクセスは，コマンド，データとも$E94003番地で受け渡しを行い，ステータスを$E94001番地で読み出します。$E94001番地への書き込みは，FDCの初期化などの非常時に使用されるコマンドに限られます。

4・1 FDCステータスレジスタ

FDCステータスレジスタの内容を図11に示します。

ビット7はCPUとFDCの間のデータ（コマンド）転送のタイミングをとるためのもので，FDCが次のデータ転送の準備ができると'1'になり，CPUがそれに応答すると'0'になります。

ビット4は，ディスクリード/ライトなどをDMAを使用せずに行うようにプログラムしたとき（SPECIFYコマンドを使用します），E-PHASE（397ページ参照）時にこのビットが'1'になり，CPUによるデータ転送要求であることを示します。C-PHASE(397ページ参照)やR-PHASE（397ページ参照）はCPUによる転送が普通ですから，このビットは意味を持たず，'0'のままになっています。

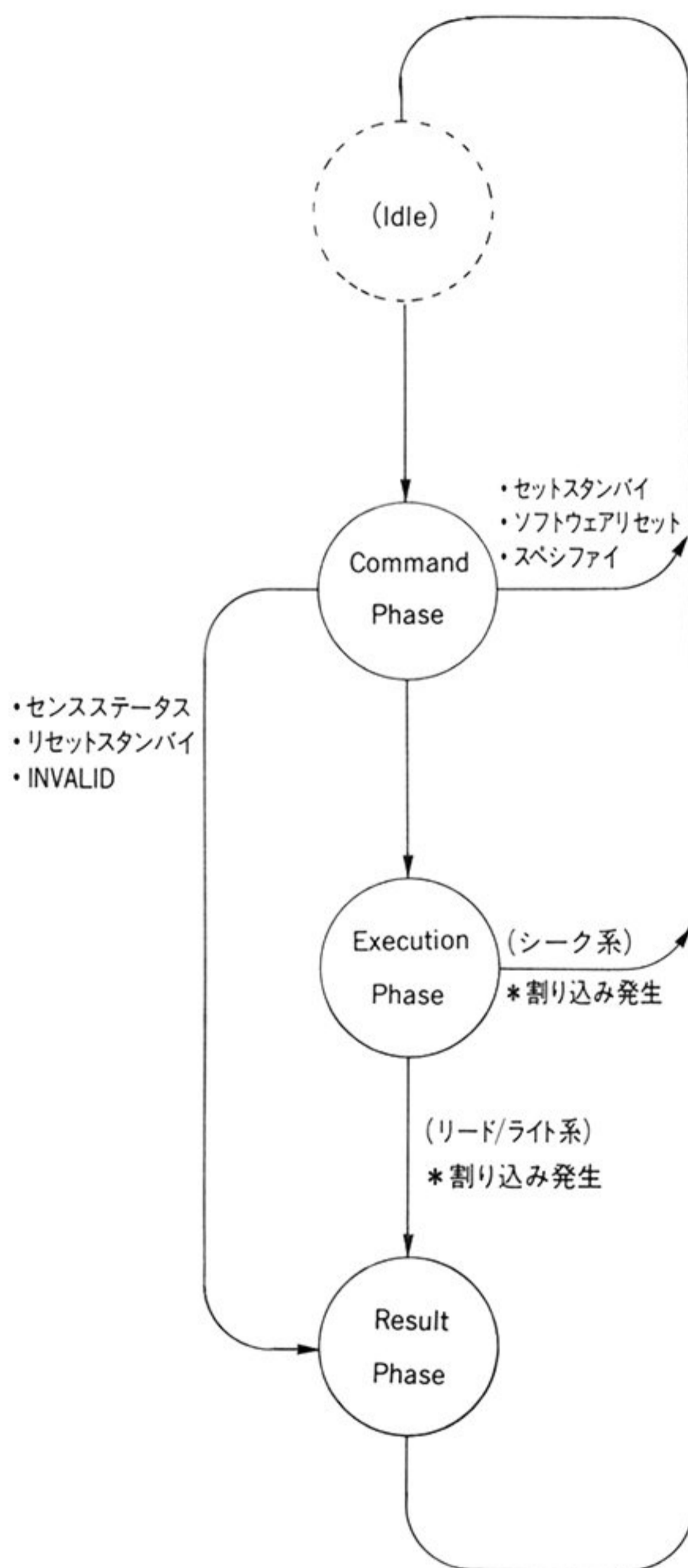
●図……11　FDCステータスレジスタ

❹·2　FDCのフェーズ遷移

FDCの動作状態は，大きく分けてCPUからコマンドや実行のためのパラメータを受け取るコマンドフェーズ（以下，C-PHASEと略します），コマンドの実行を行うエグゼキューションフェーズ（E-PHASE），実行完了ステータスをCPUが引き取るリザルトフェーズ（R-PHASE）の3つのフェーズに分類できます。398ページの図12に各コマンドごとのフェーズ遷移を図示してみましたので参考にしてください。

シーク系のコマンドやディスクリード/ライト系のコマンドの場合にはE-PHASEの完了時点で割り込みが発生します。

先頭のIdle（アイドル状態）は仮想的に考えたものです。実際にはFDCは，前回のコマンド処理が完了するとすぐに次のコマンド待ちになりますので，明確なアイドル状態は存在しないと考えることもできるのですが，フェーズ遷移を考えるうえでは，いったんアイドル状態を経由するほうが自然なので，図の中には入れておきました。

ディスクのリード/ライトなど，E-PHASEでデータ転送をともなう場合には，CPUやDMACによってデータ転送を実行します。FDDは，SASIやSCSIのハードディスクのよう

●図……12　FDCのフェーズ遷移

●図……13　1バイト分の転送時間

記録方式 / メディアタイプ	FM	MFM
2HD*1	32μs	16μs
2DD/2D*1	64μs	32μs

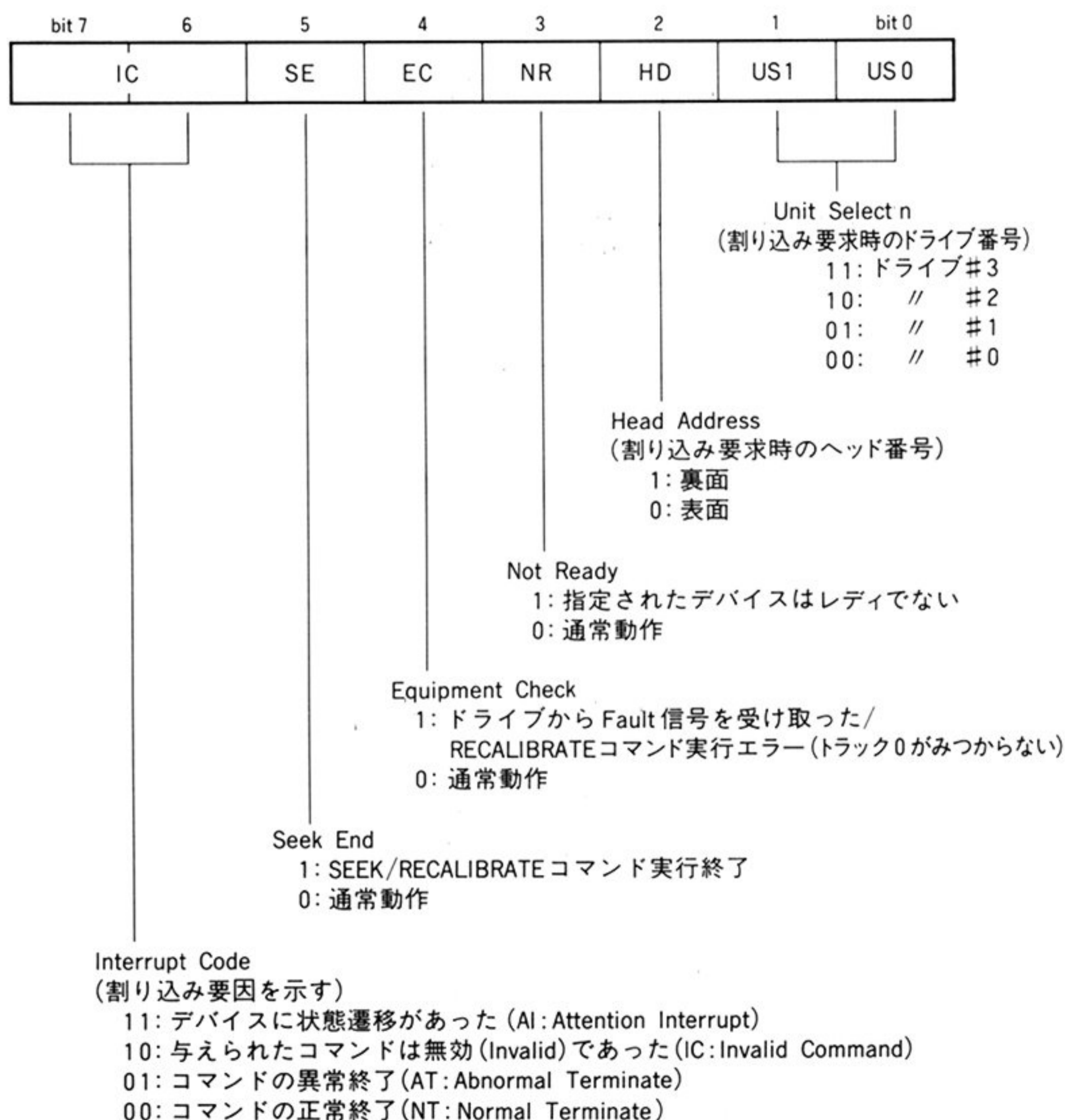
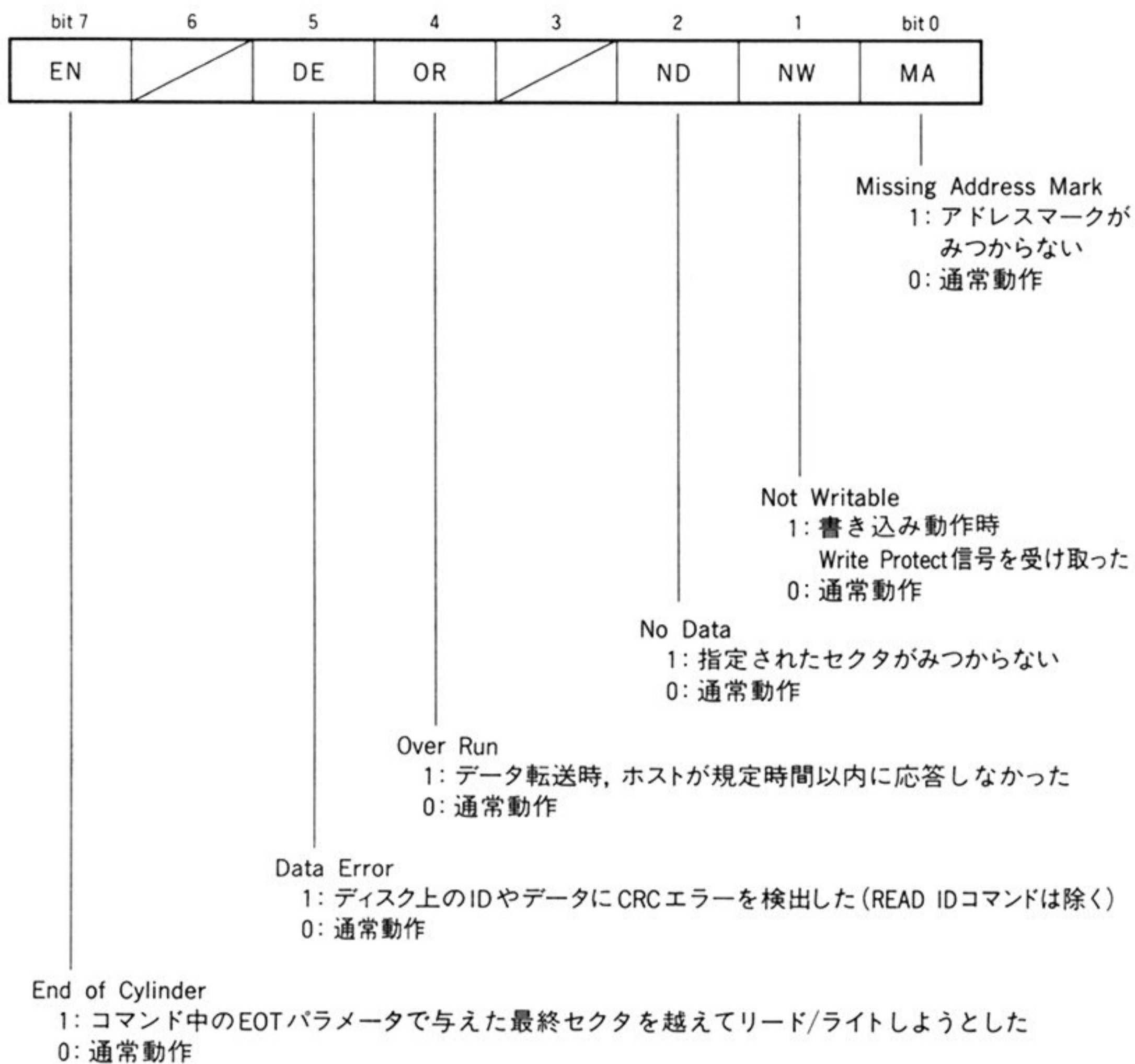
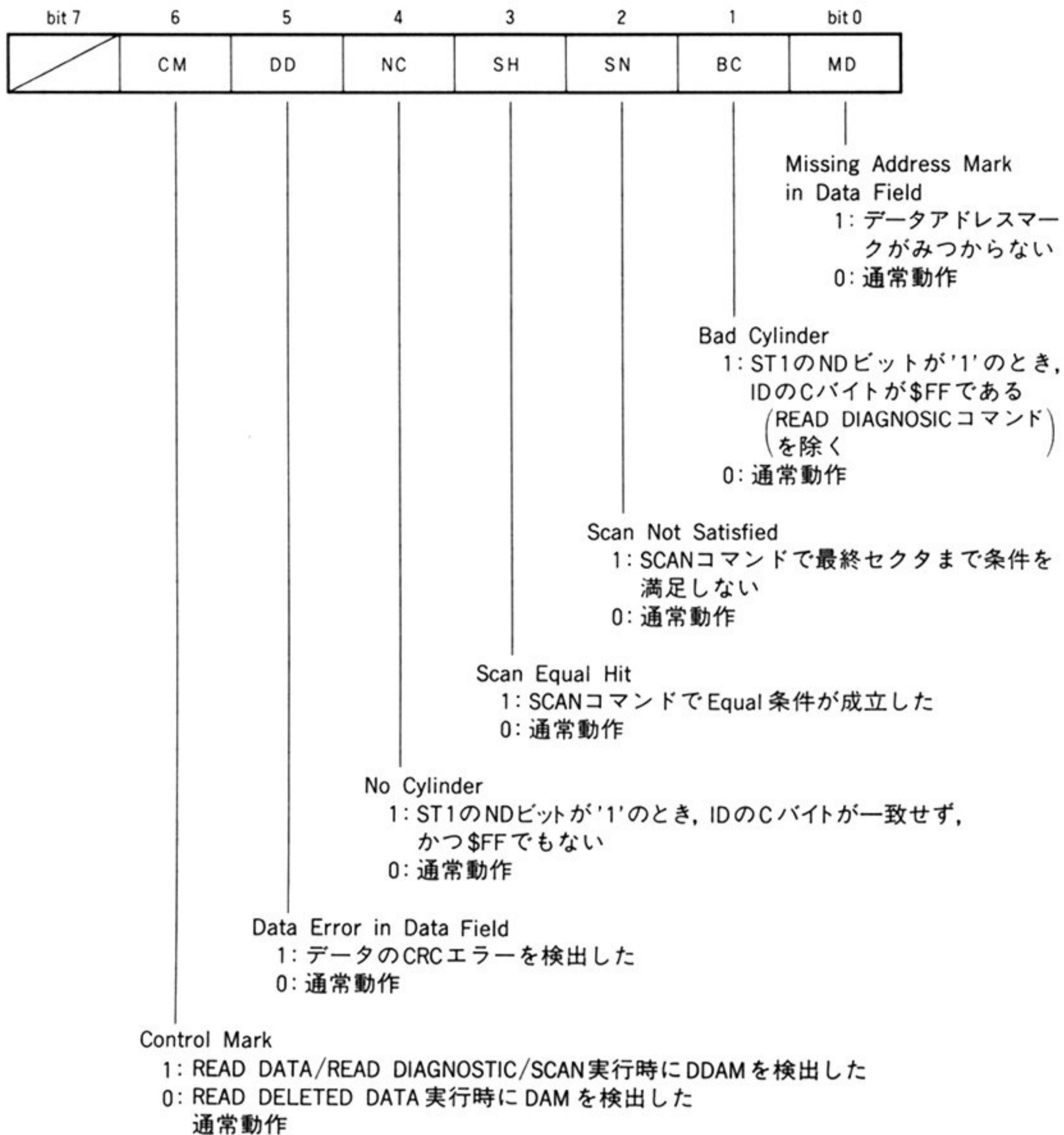
＊1：2HD/2DDになるのはMFM記録時

な大きなバッファは持っていませんので，FDCからの転送要求に対して必ず規定時間以内にサービスしなくてはなりません。このため，通常はDMACを使用してデータ転送を行います。ディスクのメディアタイプと記録方式ごとの1バイト分のデータの転送時間を図13に示します。X 68000では通常2 HDフォーマットを使用しますから，16 μs以内にデータの引き取り（リード時）や書き込み（ライト時）を完了させなくてはなりません。

④・3 リザルトステータス

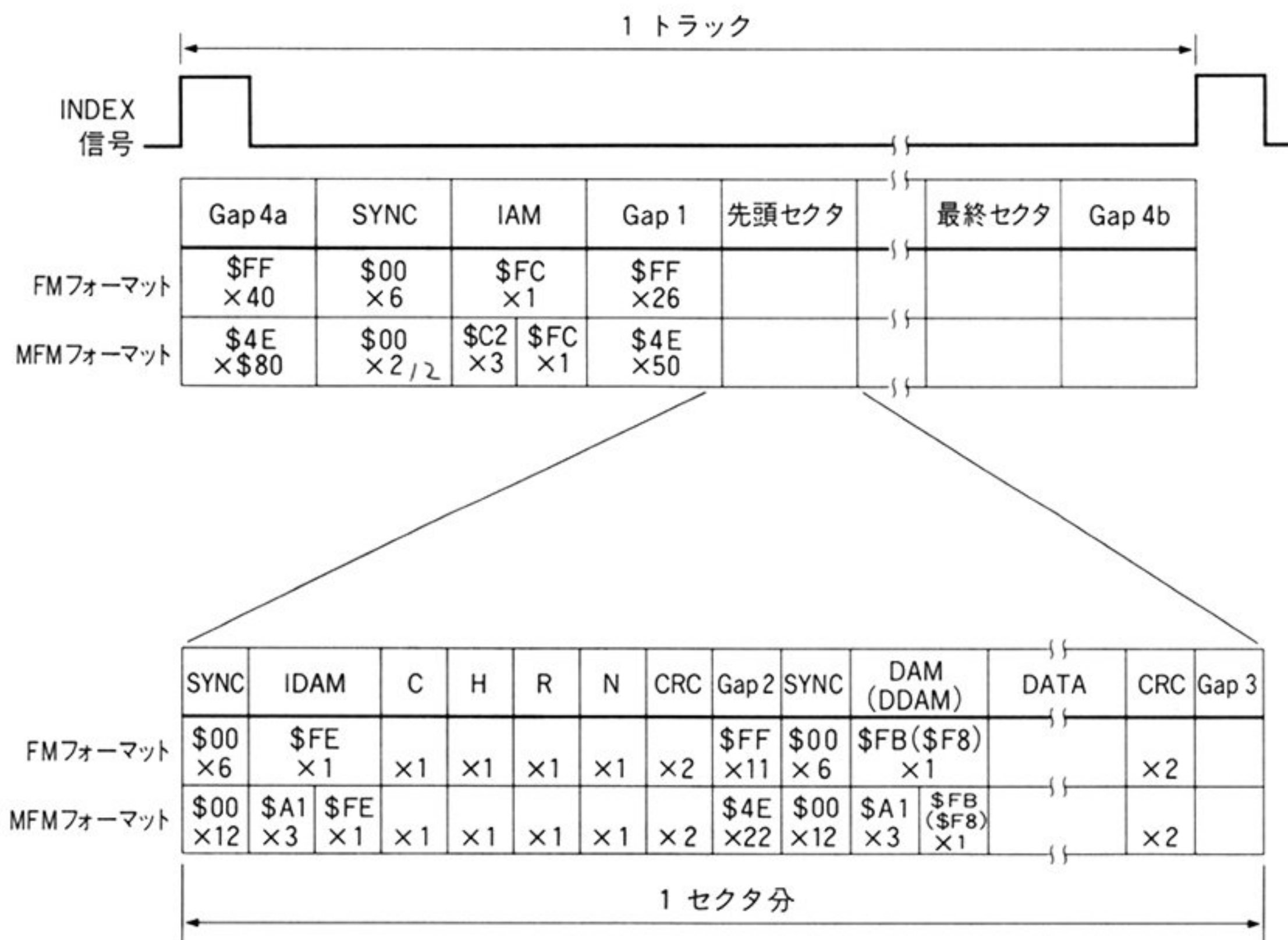
R-PHASEで返されるステータスのうち，リード/ライト系など，多くのコマンドで返されるものがST 0，ST 1，ST 2の3つのステータスバイトです。それぞれのデータのビット配置は図14，図15，図16のようになっています。

ST 0の上位2ビットが'11'のときの説明中の状態遷移というのは，ディスクの抜き挿しなどを指すのですが，X 68000では，この変化はI/Oコントローラがとらえるようになっています。

●図……14 リザルトステータス0（ST 0）

●図……15　リザルトステータス1(ST 1)

●図……16　リザルトステータス 2 (ST 2)

❹·4　トラックフォーマット

μPD 72065 が扱うディスクの 1 トラックのフォーマットの詳細を 402 ページの図 17 に示します。INDEX と書いた信号はディスクが 1 回転するごとに発生するパルス信号で，5 インチのフロッピーディスクではディスクに空けられた穴を検出して発生しています。

インデックスパルスの後には，トラックの先頭をマークする Gap 4 a，SYNC (同期パターン)，IAM (Index Address Mark)，Gap 1 と続き，その後に各セクタの情報が順に並べら

●図……17　トラックフォーマット

れます。さらに最終セクタの後にはGap 4 bがきて，1トラックの最後を示します。

各セクタの先頭は，SYNC，IDAM(ID Address Mark)に続いて，C(シリンダ)，H(ヘッド)，R(セクタ)，N(セクタ長)データが続きます。C，H，R，Nは，そのセクタが第何シリンダ（トラック）の第何セクタなのか，表面なのか，裏面のトラックなのかといったことを示すものです。セクタの住所のようなものであると考えればよいでしょう。

これらのヘッダのCRCチェックコードに続いて，Gap 2，SYNCが書き込まれ，さらにDAM (Data Address Mark)，またはDDAM (Deleted Data Address Mark) が書き込まれます。通常，この領域はDAMが書き込まれます。この領域がDDAMになっていると，通常のREAD DATA/WRITE DATAコマンドなどでアクセスしたときに，ST 2のCMビットが'1'になります。

DAM/DDAMに続いて，実際にリード/ライトを行うデータがあり，最後にこれらのCRCチェックコード，Gap 3が続きます。

●5 FDCのコマンド

FDCのコマンドは多くのパラメータを持つため，それぞれのパラメータを略称で呼ぶようになっています。FDCのコマンドの図の中で使用される略称と，その意味の対応関係の一覧を，図18に示しますので参考にしてください。

●図……18　コマンド中の略称とその意味

略称	名　称	内　容
MT	Multi track	複数トラックにわたって動作を行うときに'1'にする
MF	MFM Mode	倍密度記録を行うとき，'1'にする(X68000では通常'1'にする)
SK	Skip	DDAM/DAMのセクタをスキップさせたいとき，'1'
HD	Head	ヘッドアドレス。表='0'，裏='1'
US0, US1	Unit Select 0/1	ドライブ番号の指定（X68000では無意味）
C	Cylinder Number	ディスクのID情報中のシリンダ(トラック)番号
H	Head Number	〃　ヘッド番号
R	Record Number	〃　セクタ番号
N	Record Length	セクタ長コード
EOT	End Of Track	最終セクタ番号
GPL	Gap Length	Gap 3の書き込みバイト数
GSL	Gap Skip Length	Gap 3の読み飛ばしバイト数
DTL	DaTa Length	1セクタあたりの処理すべきバイト数(Nが$00のときだけ有効)
ST0, 1, 2	Status	リザルトステータス
SC	Sector	WRITE IDコマンド時，1トラックあたりのセクタ数を指定する
D	Data	WRITE IDコマンド時，データエリアに書き込むデータを指定する
STP	Step	SCANコマンド時，$01なら次のセクタを，$02なら1つおきに処理する
NCN	Next Cylinder Number	SEEKコマンド時，シーク先のシリンダ番号を指定する
SRT	Step Rate Time	ステップパルス(ヘッド移動信号)の間隔を指定する
HUT	Head Unload Time	ヘッドロード信号(HDLD)がOFFにしてからヘッドが離れるまでの時間(X68000では無意味)
HLT	Head Load Time	ヘッドロード信号がONにしてからヘッドが安定するまでの時間
ND	Non-DMA Mode	DMAを使わずにデータ転送をするとき，'1'にする

❺·1 READ DATAコマンド

❺·❶1 コマンドフェーズ(C-PHASE)

READ DATA コマンドのフォーマットを図 19 に示します。

C-PHASE では 9 バイトのコマンド/パラメータを与えます。それぞれのパラメータの意味(図 19 参照) は次のようになっています。

●図……19 READ DATA コマンド

フェーズ	READ/WRITE	bit 7	6	5	4	3	2	1	bit 0	備考
C-PHASE	W	MT	MF	SK	'0'	'0'	'1'	'1'	'0'	SK : Skip DDAM
							HD	US1	US0	
		C								実行開始セクタのID情報
		H								
		R								
		N								
		EOT								
		GSL								
		DTL								
E-PHASE	R									データ転送動作
R-PHASE	R	ST0								
		ST1								
		ST2								
		C								・正常終了時 実行終了セクタの次のセクタのID ・異常終了時 終了時のセクタのID
		H								
		R								
		N								

MT：マルチトラック

ディスクの表裏を連続してアクセスするとき，このビットを'1'にします。

MF：MFMモード

倍密度記録方式（2HDや2DD）を使用するとき，'1'にします。X68000では通常'1'にします。

SK：SKip DDAM

'1'にしておくと，アクセスしようとしたセクタにDDAM（Deleted Data Address Mark)が書き込まれていたとき，そのセクタへのアクセスを飛ばし，次のセクタにアクセスします。'0'になっていると，そのセクタの転送を終了した後，コマンドの実行を終了します。このとき，ST2のCM（Control Mark）ビットが'1'になります。

HD：Head

アクセスに使用するヘッド番号を指定します。'0'で表面，'1'で裏面側のヘッドの選択になります。

US0, US1：Unit Select 0/1

アクセスするドライブ番号の指定を行うものです。X68000ではドライブの選択はI/Oコントローラで行いますので，この指定はドライブ選択としての意味はありませんが，SEEKやRECALIBRATEコマンドでは，FDC内部で保存している各ドライブのシリンダ番号とのからみがありますので，実際にアクセスするドライブ番号をセットするようにしてください。

C, H, R, N：Cylinder/Head/Record/Length

アクセスするセクタのシリンダ，ヘッド番号，セクタ番号，1セクタあたりのリード/ライトする大きさを指定します。FDCはディスク上の各セクタのヘッダに書き込まれているC，H，R，N値と，ここで指定されたものを比較し，一致するとアクセスを開始します。通常，これらの値は実際のヘッドの物理的な位置やヘッド番号と一致させます。

もっとも，ここで与えるシリンダ番号やヘッド番号はたんにセクタに書き込んである値と比較するためのものですから，たとえば，意図的にディスク上のすべてのセクタのCを0にしたようなディスクを作成して，読み出し時にCをすべて0として読み出すといったようなこともできないことではありません

EOT：End Of Track

トラック中の最終セクタの番号をセットします。

GSL：Gap SKip Length

複数のセクタを連続アクセスするマルチセクタ動作のときに，データ部と次のセクタのID部の不連続領域を読み飛ばすためのものです。

DTL：DaTa Length

Nの値が$00のとき，このデータで1セクタあたりアクセスするデータ長を指定します。Nが

$00 以外のときにはこのデータは意味を持ちません。

図 20 によく使用されるフォーマットごとの各パラメータの設定値を示しますので参考にしてください。なお，Human 68 K では MFM で 1024 バイト/セクタのフォーマットを採用しています。

●図……20　セクタフォーマットとパラメータ

フォーマット		FDCに与えるパラメータ				備　　考
記録方式	セクタサイズ (バイト/セクタ)	N	EOT, SC	GSL	GPL	
FM	128	$00	$1A	$07	$1B	(IBMディスケット 1)
	256	$01	$0F	$0E	$2A	(IBMディスケット 2)
	512	$02	$08	$1B	$3A	
	1024	$03	$04	未定	未定	
	2048	$04	$02	未定	未定	
	4096	$05	$01	未定	未定	
MFM	256	$01	$1A	$0E	$36	(IBMディスケット 2D)
	512	$02	$0F	$1B	$54	
	1024	$03	$08	$35	$74	X68000 標準フォーマット (IBMディスケット 2D)
	2048	$04	$04	未定	未定	
	4096	$05	$02	未定	未定	
	8192	$06	$01	未定	未定	

(　　)内は，8インチFDでのフォーマット名称

❺·❶ 2　エグゼキューションフェーズ(E-PHASE)

データ転送を実行します。データ転送を DMA で行う場合には，コマンドを与え終わる前に DMA のセットアップをしておいたほうがよいでしょう。

アクセスの終了は，TC (ターミナルカウント) 端子によって伝えられます。TC で終了が通知されないかぎり，FDC は次のセクタの読み出しを連続して行っていきます。Nを$00 とし，DTL を 128 未満にしたときは，各セクタの先頭から DTL バイトずつをつまみ食いするようにして転送が行われていきます。

❺・❶ 3 リザルトフェーズ（R-PHASE）

R-PHASEでは，ST 0, ST 1, ST 2の3つのリザルトステータスと，終了時のC，H，R，N値が返されます。C，H，R，Nは，正常終了したときには最後にアクセスしたセクタの次のセクタのIDが，異常終了したときには終了時のセクタのIDが返されます。

❺・2 READ DELETED DATAコマンド

コマンドのフォーマットを図21に示します。セクタのヘッダ中，データの前がDDAM (Deleted Data Address Mark) になっているセクタを読み出すコマンドです。動作は，

●図……21 READ DELETED DATA コマンド

フェーズ	READ/WRITE	bit7	6	5	4	3	2	1	bit0	備考
C-PHASE	W	MT	MF	SK	'0'	'1'	'1'	'0'	'0'	SK:Skip DAM
							HD	US1	US0	
		C								実行開始セクタのID情報
		H								
		R								
		N								
		EOT								
		GSL								
		DTL								
E-PHASE	R									データ転送動作
R-PHASE	R	ST0								
		ST1								
		ST2								
		C								・正常終了時 実行終了セクタの次のセクタのID ・異常終了時 終了時のセクタのID
		H								
		R								
		N								

READ DATA コマンドの説明中の DAM を DDAM に，DDAM を DAM に入れ替えたものに相当します。

たとえば，READ DATA コマンドでは，DDAM になっているセクタを読み出すと，ST 2 の CM ビットを '1' にしましたが，このコマンドでは DAM になっているセクタを読み出すと '1' になります。

❺·3 READ IDコマンド

コマンドフォーマットを図 22 に示します。コマンドを受け取ってから最初に見つけた正常な（エラーのない）セクタの ID 情報（C，H，R，N）を取り込みます。このコマンドの E-PHASE では FDC がディスクから ID を取り込むだけで，ホスト（CPU/DMA）との間でのデータ転送は行われません。

●図……22 READ ID コマンド

フェーズ	READ/WRITE	bit7	6	5	4	3	2	1	bit0	備考
C-PHASE	W	'0'	MF	'0'	'0'	'1'	'0'	'1'	'0'	
							HD	US1	US0	
E-PHASE	—									エラーのないID情報を見つける
R-PHASE	R	ST0								
		ST1								
		ST2								
		C								E-PHASEで読み取ったID情報
		H								
		R								
		N								

❺·4 WRITE IDコマンド

1 トラック分のフォーマットを行います。コマンドフォーマットは図 23 のようになっています。SC，GPL の設定値は READ DATA のところの図を参照してください。D バイトは，

●図……23　WRITE ID コマンド

フェーズ	READ/WRITE	bit 7	6	5	4	3	2	1	bit 0	備　　考
C-PHASE	R	'0'	MF	'0'	'0'	'1'	'1'	'0'	'1'	
							HD	US1	US0	
		N								
		SC								
		GPL								
		D								
E-PHASE	W									1トラック分のID情報の転送
R-PHASE	R	ST0								
		ST1								
		ST2								
		C								無意味
		H								
		R								
		N								C-PHASEで与えた値

各セクタのデータ部分に書き込む値を指定します。

E-PHASEで与えるのは，各セクタのID(C，H，R，N)です。つまり，WRITE IDコマンドのE-PHASEでFDCに転送するデータは，4×(トラックあたりのセクタ数)となります。

R-PHASEで戻ってくる値のうち，C，H，Rは意味を持ちません。NバイトはC-PHASEで与えたNの値がそのまま返されます。

❺·5　WRITE DATAコマンド

コマンドフォーマットを410ページの図24に示します。指定したセクタにデータの書き込みを行います。E-PHASEのデータ転送方向が逆になるほかはREAD DATAコマンドと変わるところはありません。

●図……24　WRITE DATA コマンド

フェーズ	READ/WRITE	bit7	6	5	4	3	2	1	bit0	備　考
C-PHASE	W	MT	MF	'0'	'0'	'0'	'1'	'0'	'1'	
							HD	US1	US0	
		C								実行開始セクタのID情報
		H								
		R								
		N								
		EOT								
		GSL								
		DTL								
E-PHASE	W									データ転送
R-PHASE	R	ST0								
		ST1								
		ST2								
		C								・正常終了時 実行終了セクタの次のセクタのID ・異常終了時 終了時のセクタのID
		H								
		R								
		N								

❺·6 WRITE DELETED DATAコマンド

コマンドフォーマットを図 25 に示します。セクタのヘッダ中に DAM のかわりに DDAM を書き込むほかは WRITE DATA コマンドと同様です。

●図……25　WRITE DELETED DATAコマンド

フェーズ	READ/WRITE	bit7	6	5	4	3	2	1	bit0	備　　考
C-PHASE	W	MT	MF	'0'	'0'	'1'	'0'	'0'	'1'	
							HD	US1	US0	
		C								実行開始セクタのID情報
		H								
		R								
		N								
		EOT								
		GSL								
		DTL								
E-PHASE	W									データ転送
R-PHASE	R	ST0								
		ST1								
		ST2								
		C								・正常終了時 実行終了時のセクタの次のセクタID ・異常終了時 実行終了時のセクタID
		H								
		R								
		N								

❺·7　READ DIAGNOSTICコマンド

コマンドフォーマットは412ページの図26のようになっています。READ DATAコマンドとよく似ているのですが，INDEX信号の直後のセクタからエラーの有無に関係なく強制的に読み出していく点が異なります。コマンド中のC，H，R，Nがセクタのものと一致しなくても，ST 1のND (No Data) ビットを'1'にするだけで処理を継続し，正常終了します。

IDやデータのCRCエラーがあっても，ST 1のDE (Data Error) やST 2のDD (Data Error in Data Field) ビットを'1'にするだけで正常終了します。

DDAMを検出すると，ST 2のCM (Control Mark) ビットを'1'にしますが，処理は継続します。

●図……26　READ DIAGNOSTIC コマンド

フェーズ	READ/WRITE	bit 7	6	5	4	3	2	1	bit 0	備　　考
C-PHASE	W	'0'	MF	'0'	'0'	'0'	'0'	'1'	'0'	
							HD	US1	US0	
		C								実行開始セクタのID情報
		H								
		R								
		N								
		EOT								
		GSL								
		DTL								
E-PHASE	R									データ転送
R-PHASE	R	ST0								
		ST1								
		ST2								
		C								・正常終了時 実行終了時のセクタの次のセクタID ・異常終了時 実行終了時のセクタID
		H								
		R								
		N								

コマンド終了時に読み取れるエラーステータスは，処理中に起きたすべてのエラー条件の論理和となっています。

❺·8　SCAN EQUAL/SCAN LOW OR EQUAL/SCAN HIGH OR EQUALコマンド

各コマンドのフォーマットを図27，図28，図29に示します。指定したセクタの内容と，ホストからFDCに書き込むデータの比較を行います。データはセクタの先頭データから順に比較されていきます。ただし，ホストからFDCに書き込んだデータが$FFであるときは，そのバイトの比較は行われず，等しいものとして扱われます。

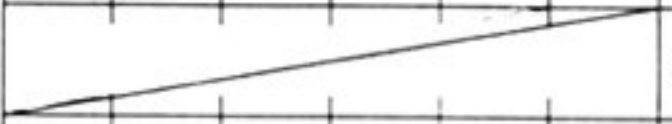
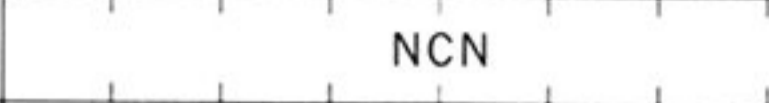
●図……27　SCAN EQUAL コマンド

フェーズ	READ/WRITE	bit 7	6	5	4	3	2	1	bit 0	備　考
C-PHASE	W	MT	MF	SK	'1'	'0'	'0'	'0'	'1'	
							HD	US1	US0	
		C								実行開始セクタのID情報
		H								
		R								
		N								
		EOT								
		GSL								
		STP								
E-PHASE	W									データ比較
R-PHASE	R	ST0								
		ST1								
		ST2								
		C								比較最終セクタ
		H								
		R								
		N								

SCAN EQUAL は比較したセクタの内容がすべて等しいときに，SCAN LOW OR EQUAL はすべてが一致するか，一致しなかった最初のデータを比べたときに，ディスクから読み取ったほうが小さいとき，SCAN HIGH OR EQUAL はすべてが一致するか，最初に一致しなかったデータを比べたときに，ディスクから読み取ったデータが大きかったときに正常終了します。

●図……28　SCAN LOW OR EQUAL コマンド

フェーズ	READ/WRITE	bit7	6	5	4	3	2	1	bit0	備考
C-PHASE	W	MT	MF	SK	'1'	'1'	'0'	'0'	'1'	
							HD	US1	US0	
		C								実行開始セクタのID情報
		H								
		R								
		N								
		EOT								
		GSL								
		STP								
E-PHASE	W									データ比較
R-PHASE	R	ST0								
		ST1								
		ST2								
		C								最終比較セクタのID情報
		H								
		R								
		N								

●図……29　SCAN HIGH OR EQUAL コマンド

フェーズ	READ/WRITE	bit 7	6	5	4	3	2	1	bit 0	備考
C-PHASE	W	MT	MF	SK	'1'	'1'	'1'	'0'	'1'	
							HD	US1	US0	
		C								実行開始セクタのID情報
		H								
		R								
		N								
		EOT								
		GSL								
		STP								
E-PHASE	W									データ比較
R-PHASE	R	ST0								
		ST1								
		ST2								
		C								最終比較セクタのID情報
		H								
		R								
		N								

❺·9　SEEKコマンド

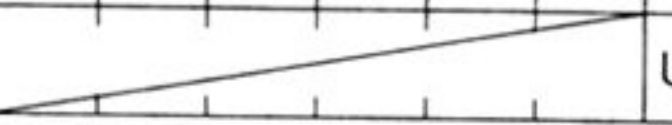
コマンドフォーマットは416ページの図30のようになっています。ヘッドを指定したシリンダに移動します。FDCは前回のヘッド位置を覚えており，ヘッドの移動方向の管理や移動する量はFDCが自動的に判断し，ヘッドを移動させますので，CPUがコマンドで与えるのは，目的のシリンダ番号だけですむようになっています。

なんらかの原因で，FDCが管理しているシリンダ番号と実際のヘッド位置が不一致になったような場合は，リード/ライトなどを行ったときにコマンド中のIDとセクタのIDが一致しないため，エラーとなります。このようなときはに次に説明するRECALIBRATEコマンドを使って，FDCの管理しているシリンダ番号と実際のドライブのヘッド位置をともに0に初期化します。

●図……30　SEEK コマンド

フェーズ	READ/WRITE	bit 7	6	5	4	3	2	1	bit 0	備　考
C-PHASE	W	'0'	'0'	'0'	'0'	'1'	'1'	'1'	'1'	
								US1	US0	
		NCN								
E-PHASE	—									シーク動作実行

SEEK 動作の終了は割り込みで通知されます。割り込みが発生したとき，FDC ステータスレジスタの DIO ビットが '0' になっていることを確認したら，SENSE INTERRUPT STATUS コマンドを送って ST 0 を引き取ります。

❺·10　RECALIBRATEコマンド

コマンドフォーマットは図 31 のようになっています。FDC 内部で管理しているシリンダ番号と実際の FDD のヘッド位置をともに 0 にします。SEEK コマンドと似ていますが，SEEK は FDC が内部で管理しているシリンダ番号と，与えられたシリンダ番号の差分のヘッド移動を行うものであるのに対し，RECALIBRATE コマンドは，FDD がハード的に持っている 0 トラック検出機構から出力される信号 (TRK 00) を使用し，この信号が出力されるまでヘッドを移動させていく点が異なります。

FDD を使用しはじめるときや，ディスクのリードエラーが起きたときなどに，ヘッド位置を初期化し，FDC の管理しているシリンダ位置と一致させるために使用されます。

このコマンドは FDD へのアクセスだけで，ディスクへのアクセスをともなわないため，X 68000 では FDD が接続されているか否かのチェックに使用しているというのは前にも述べたとおりです。

RECALIBRATE コマンドの動作の終了も割り込みで通知されます。割り込みが発生した

●図……31　RECALIBRATE コマンド

フェーズ	READ/WRITE	bit 7	6	5	4	3	2	1	bit 0	備　考
C-PHASE	W	'0'	'0'	'0'	'0'	'0'	'1'	'1'	'1'	
								US1	US0	
E-PHASE	—									リキャリブレート動作

とき，FDC ステータスレジスタの DIO ビットが'0'になっていることを確認したら，SENSE INTERRUPT STATUS コマンドを送って ST 0 を引き取ります。

❺·11 SENSE INTERRUPT STATUSコマンド

コマンドフォーマットは図 32 のようになっています。割り込みが発生したとき，FDC ステータスレジスタの DIO ビットが'0'になっていた場合，このコマンドを発行して ST 0 とコマンド終了時のヘッドのシリンダ位置を引き取ります。

●図……32 SENSE INTERRUPT STATUS コマンド

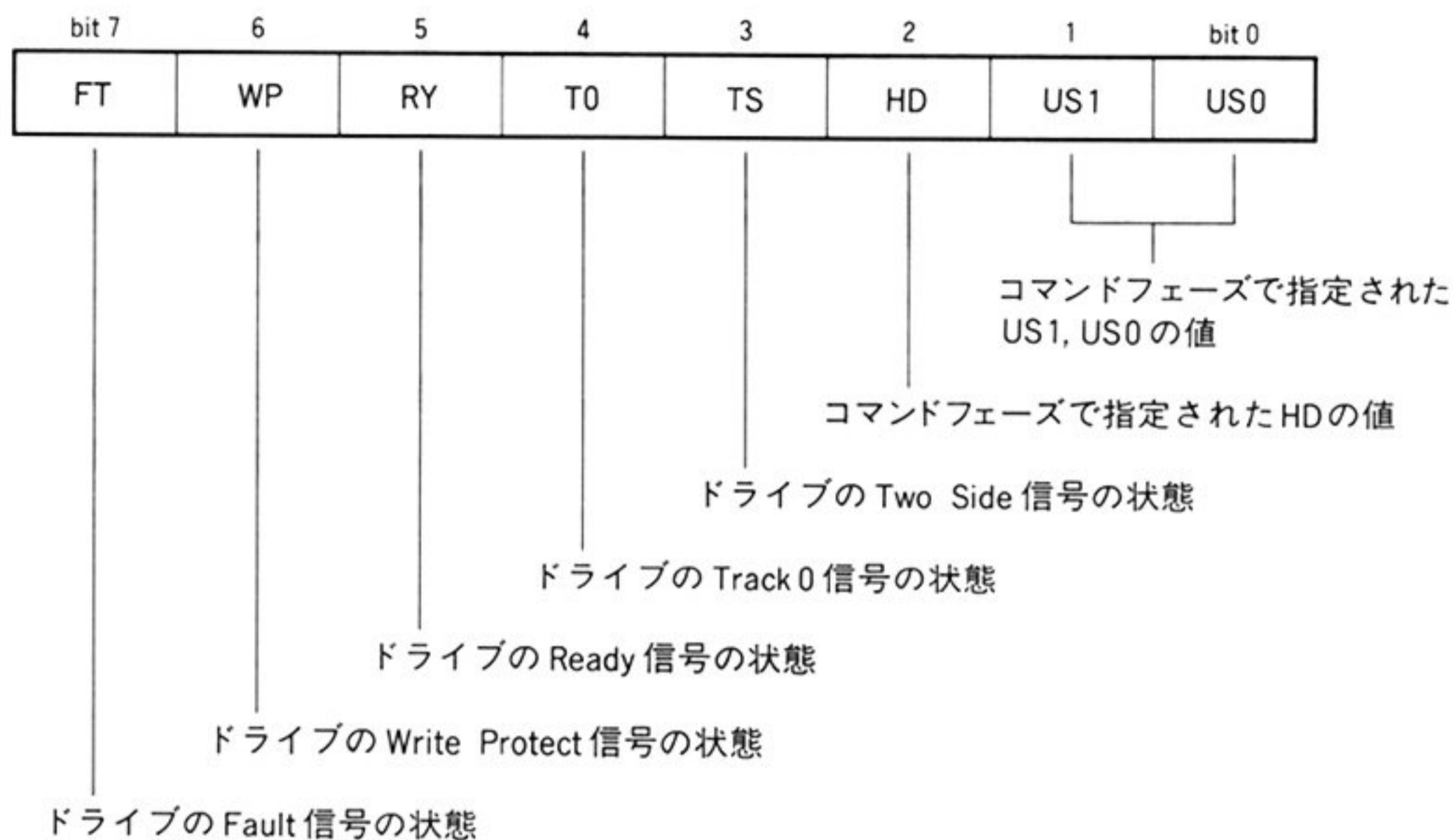
フェーズ	READ/WRITE	bit 7	6	5	4	3	2	1	bit 0	備考
C-PHASE	W	'0'	'0'	'0'	'0'	'1'	'0'	'0'	'0'	
E-PHASE	R	ST 0								
		PCN								コマンド終了時のシリンダ位置

❺·12 SENSE DEVICE STATUSコマンド

コマンドフォーマットは図 33 のようになっています。このコマンドを使うと，FDC に入力される FDD のステータスラインの状態が読み出されます。リザルトステータスで引き取られるステータス（ST 3）の内容は 418 ページの図 34 のようになっています。

●図……33 SENSE DEVICE STATUS コマンド

フェーズ	READ/WRITE	bit7	6	5	4	3	2	1	bit 0	備考
C-PHASE	W	'0'	'0'	'0'	'0'	'0'	'1'	'0'	'0'	
							HD	US1	US0	
R-PHASE	R	ST 3								ドライブの状態

●図……34　リザルトステータス3（ST 3）

❺・13　SPECIFYコマンド

ディスクドライブの種別ごとに設定が必要なパラメータの設定を行います。コマンドフォーマットは図35のようになっています。

SRT (STEP RATE TIME) は，ステップパルス（ヘッド移動信号）の間隔を設定するものです。X 68000の内蔵ドライブの場合は3 msです。

HUT (HEAD UNLOAD TIME) は，ディスクのリード/ライト系コマンドが終了してから，ヘッドをアンロード状態（ディスクから離れた状態）にするまでの時間を設定します。この時間以内に再度アクセスがなかった場合には，次のリード/ライト系コマンドが発行されてから実際のアクセス開始までHLTで指定した時間がとられることになります。

HLT (HEAD LOAD TIME) は，ディスクのリード/ライト系コマンドの実行開始時，ヘッドがロード状態（ディスクに接触し，安定した状態）になるまでの待ち時間を設定します。

●図……35　SPECIFY コマンド

フェーズ	READ/WRITE	bit 7	6	5	4	3	2	1	bit 0	備　考
C-PHASE	W	'0'	'0'	'0'	'0'	'0'	'0'	'1'	'1'	
		SRT				HUT				
		HLT							ND	

8インチタイプのFDDや，5インチタイプでも厚型のものでは，ヘッドが物理的にディスクと接触/離脱動作をするようになっており，名前のとおり，ロード/アンロードだったのですが，X 68000の内蔵ドライブではヘッドはつねにディスクと接触状態にありますので，これらのパラメータは名目だけです。HUTを長めにしておくと，少し間をあけながら連続してアクセスするとき，HLT分の時間が不要になりますので，アクセスが若干高速になります。

NDビットはリード/ライト系コマンドのE-PHASEのデータ転送をCPUで行うか，DMAで行うかを設定するものです。'1'にするとCPUによる転送，'0'にするとDMAによる転送になります。X 68000では，とくに理由のないかぎり，DMAモードで使うのが普通でしょう。

SRT，HUT，HLTの各パラメータの設定値とそれぞれの時間の関係は図36のようになっ

●図……36　SRT, HUT, HLTの設定値と時間

単位:ms

設定値	2HD			2DD/2D		
	SRT	HUT	HLT	SRT	HUT	HLT
$00	16	禁止	禁止	32	禁止	禁止
$01	15	16	2	30	32	4
$02	14	32	4	28	64	8
$03	13	48	6	26	96	12
$04	12	64	8	24	128	16
$05	11	80	10	22	160	20
$06	10	96	12	20	192	24
$07	9	112	14	18	224	28
$08	8	128	16	16	256	32
$09	7	144	18	14	288	36
$0A	6	160	20	12	320	40
$0B	5	176	22	10	352	44
$0C	4	192	24	8	384	48
$0D	3	208	26	6	416	52
$0E	2	224	28	4	448	56
$0F	1	240	30	2	480	60
$10			32			64
$11			34			68
⋮						
$7E			252			504
$7F			254			508

ています。

❺·14 SET STANDBYコマンド

コマンドは図37のようになっています。

SET STANDBYコマンドは，FDCの内部クロックを停止させ，スタンバイ状態にします。このコマンドにはE-PHASEもR-PHASEもなく，書き込んでから約3μs後にスタンバイ状態に移行します。この状態でもFDCの内部状態や出力端子の状態は保持されます。クロックが停止するため，消費電流は少なくなりますが，X68000のようにAC電源で動いているようなものの場合にはあまり意味がないコマンドです。

●図……37 SET STANDBYコマンド

フェーズ	READ/WRITE	bit 7	6	5	4	3	2	1	bit 0	備考
C-PHASE	W	'0'	'0'	'1'	'1'	'0'	'1'	'0'	'1'	

❺·15 RESTE STANDBYコマンド

スタンバイ状態を解除します。コマンドは図38のようになっています。このコマンドは，FDC内部ではINVALID (無効) コマンドと同じ扱いであり，R-PHASEでST 0を引き取る必要があります。

●図……38 RESET STANDBYコマンド

フェーズ	READ/WRITE	bit 7	6	5	4	3	2	1	bit 0	備考
C-PHASE	W	'0'	'0'	'1'	'1'	'0'	'1'	'0'	'0'	
R-PHASE	R	ST0								$80が返ってくる

❺·16 SOFTWARE RESETコマンド

コマンドは図39のようになっています。このコマンドはFDCをリセット（初期化）し，ハード的なリセットがかかったのと同じ状態にします。このコマンドは任意のタイミングで与えることができます。

●図……39　SOFTWARE RESET コマンド

フェーズ	READ/WRITE	bit 7	6	5	4	3	2	1	bit 0	備　考
C-PHASE	W	'0'	'0'	'1'	'1'	'0'	'1'	'1'	'0'	

❺·17 FDCパラメータ/ステータス一覧

FDCのコマンドに付随するパラメータと，R-PHASEで受け取るステータスの一覧を422ページの図40にまとめてみました。パラメータは図の左側から右に順に○印のついているものを送り，ステータスは同じく左から右に○印のついているものを引き取っていくようにします。

●図……40　FDCパラメータ/ステータス一覧

コマンド名	C-PHASE時のパラメータ														E-PHASE	R-PHASE								
	C	H	R	N	EOT	SC	GSL	GPL	DTL	D	STP	NCN	SRT/HUT	HLT/ND	READ/WRITE	STO	ST1	ST2	C	H	R	N	PCN	ST3
READ DATA	○	○	○	○	○		○		○						R	○	○	○	○	○	○	○		
READ DELETED DATA	○	○	○	○	○		○		○						R	○	○	○	○	○	○	○		
READ ID															—	○	○	○	○	○	○	○		
WRITE ID				○		○		○		○					W									
WRITE DATA	○	○	○	○	○		○		○						W	○	○	○	○	○	○	○		
WRITE DELETED DATA	○	○	○	○	○		○		○						W	○	○	○	○	○	○	○		
READ DIAGNOSTIC	○	○	○	○	○		○		○						R	○	○	○	○	○	○	○		
SCAN EQUAL	○	○	○	○	○		○				○				W	○	○	○	○	○	○	○		
SCAN LOW OR EQUAL	○	○	○	○	○		○				○				W	○	○	○	○	○	○	○		
SCAN HIGH OR EQUAL	○	○	○	○	○		○				○				W	○	○	○	○	○	○	○		
SEEK												○			—									
RECALIBRATE															—									
SENSE INTER-RUPT STATUS															—	○							○	
SENSE DEVICE STATUS															—									○
SPECIFY													○	○	—									
SET STANDBY															—									
RESET STANDBY															—	○								
SOFTWARE RESET															—									

6 サンプルプログラム

FDCアクセスのサンプルプログラムとして，フロッピーディスクの読み取りを行うプログラムを作成してみました。パラメータでブロック番号を与えると，ブロック番号をトラック(シリンダ)，ヘッド，セクタの各番号に変換して読み出しを行います。

アクセスを開始したときにモータの回転が停止していると，起動時間（約0.5s）だけ待たないとドライブがレディ状態にならないため，このプログラムではドライブのREADY信号が'1'になるまでSENSE DRIVE STATUSコマンドを繰り返し送り続けるようにしています。

また，このプログラムでは割り込みを使用しないため，FDC割り込みを禁止しています。これを行っておかないと，割り込みが発生したとたんにHuman 68KのFDC割り込み処理が行われてしまい，つじつまがあわなくなってしまいます。

●リスト……1　フロッピーディスク読み込み

```
/*
 *  ＦＤＣアクセステスト
 * XC ではvolatile がサポートされていないため、
 * 次の一行を入れてvolatileを無効にしてください
 * #define  volatile
 */

#include <doslib.h>

struct  DMAREG {
    unsigned char   csr;
    unsigned char   cer;
    unsigned short  spare1;
    unsigned char   dcr;
    unsigned char   ocr;
    unsigned char   scr;
    unsigned char   ccr;
    unsigned short  spare2;
    unsigned short  mtc;
    unsigned char   *mar;
    unsigned long   spare3;
    unsigned char   *dar;
    unsigned short  spare4;
```

```
    unsigned short  btc;
    unsigned char   *bar;
    unsigned long   spare5;
    unsigned char   spare6;
    unsigned char   niv;
    unsigned char   spare7;
    unsigned char   eiv;
    unsigned char   spare8;
    unsigned char   mfc;
    unsigned short  spare9;
    unsigned char   spare10;
    unsigned char   cpr;
    unsigned short  spare11;
    unsigned char   spare12;
    unsigned char   dfc;
    unsigned long   spare13;
    unsigned short  spare14;
    unsigned char   spare15;
    unsigned char   bfc;
    unsigned long   spare16;
    unsigned char   spare17;
    unsigned char   gcr;
} ;
volatile struct DMAREG  *dma;

volatile unsigned char  *fdc_stat = (unsigned char *)0xe94001;
volatile unsigned char  *fdc_data = (unsigned char *)0xe94003;
volatile unsigned char  *fdd_sel  = (unsigned char *)0xe94007;
volatile unsigned char  *int_stat = (unsigned char *)0xe9c001;

#define BUFSIZE 0x400
unsigned char   diskbuf[BUFSIZE];


void main();
void fd_wait_ready();
void fd_seek();
void motor_on();
void motor_off();
unsigned int fdc_sense_int_stat();
void fdc_int_mask();
void fdc_send_command();
void fdc_read_status();
void fdc_send();
unsigned int fdc_read();
```

```
void dma_setup();
void dma_start();
void dma_stop();
void wait_complete();
void clear_flag();

void main(argc,argv)
    int argc;
    char    *argv[];
{
    unsigned int    i,j,block,track,sector,head;
    unsigned char   c;
    if (argc < 2)
        block = 0;
    else    block = atoi(argv[1]);
    printf("block # = %d¥n",block);
    printf(" Track  = %d¥n",track  = block >> 4);
    printf(" Head   = %d¥n",head   = (block & 0x8) >> 3);
    printf(" Sector = %d¥n",sector = (block & 0x7)+1);
    SUPER(0);
    fdc_int_mask();
    printf("Motor ON!¥n");
    motor_on();
    printf("Wait Ready!¥n");
    fd_wait_ready();
    printf("SEEK!¥n");
    fd_seek(track);
    dma = (struct DMAREG *)0xe84000;
    clear_flag();
    dma_setup();
    dma_start();
    printf("READ DATA!¥n");
    fdc_send_command(track, head, sector);
    printf("Wait Complete!¥n");
    wait_complete();

    printf("Read Status =");
    fdc_read_status();
    for (i=0; i<BUFSIZE; i+=0x10) {
        for (j=0; j<0x10; j++)
            printf("%02X ",diskbuf[i+j]);
        for (j=0; j<0x10; j++) {
            c = diskbuf[i+j];
            if ((c < 0x20) || (c >= 0xe0) || ((c >= 0x80) && (c < 0xa0)))
                printf(".");
            else    printf("%c",diskbuf[i+j]);
        }
```

```
        printf("¥n");
    }
    motor_off();
    fdc_int_umask();
}

void fd_wait_ready()
{
    do {
        fdc_send(0x04);
        fdc_send(0x00);
    } while((fdc_read() & 0x20) == 0);
}

void fd_seek(track)
    unsigned int    track;
{
    fdc_send(0x0f);
    fdc_send(0x00);
    fdc_send(track);
    fdc_sense_int_stat();
}

unsigned int fdc_sense_int_stat()
{
    unsigned int    stat;
    while(!(*int_stat & 0x80))
        ;
    fdc_send(0x08);
    printf("Interrupt Status = ");
    printf("%02X ",stat = fdc_read());
    printf("%02X¥n",fdc_read());
    return(stat);
}

void motor_on()
{
    *fdd_sel = 0x80;
}

void motor_off()
{
    *fdd_sel = 0x00;
}
```

```
void fdc_int_mask()
{
    *int_stat &= 0xfb;
}

fdc_int_umask()
{
    *int_stat |= 0x4;
}

void fdc_send_command(trk, head, sect)
    unsigned int    trk, head, sect;
{
    fdc_send(0x46);      /* Command      */
    fdc_send(head <<2);  /* HD/US1/US0        */
    fdc_send(trk);       /* Cylinder     */
    fdc_send(head);      /* Head         */
    fdc_send(sect);      /* Record(Sector)    */
    fdc_send(0x03);      /* Num(Block Length)     */
    fdc_send(0x08);      /* EOT          */
    fdc_send(0x35);      /* GSL          */
    fdc_send(0x00);      /* DTL(Not Used)     */
}

void fdc_read_status()
{
    unsigned int    i;
    for (i = 0; i<0x7; i++)
        printf(" %02X", fdc_read());
    printf("¥n");
}

void fdc_send(dat)
    unsigned int    dat;
{
    unsigned int    stat;
    printf("Send:-%02X¥n", dat);
    while((*fdc_stat & 0xc0) != 0x80)
        ;
    *fdc_data = dat;
}

unsigned int fdc_read()
{
    while((*fdc_stat & 0xc0) != 0xc0)
        ;
```

```
    return(*fdc_data);
}

void dma_setup()
{
    dma->dcr = 0x80;
    dma->ocr = 0xb2;
    dma->scr = 0x04;
    dma->ccr = 0x00;
    dma->cpr = 0x08;
    dma->mfc = 0x05;
    dma->dfc = 0x05;

    dma->mtc = BUFSIZE;
    dma->mar = diskbuf;
    dma->dar = (unsigned char *)fdc_data;
}

void dma_start()
{
    dma->ccr |= 0x80;
}

void wait_complete()
{
    while(!(dma->csr & 0x90))
        ;
}

void clear_flag()
{
    dma->csr = 0xff;
}
```

SASI

初代機以来，ハードディスクインタフェースとして採用されてきたSASIインタフェースは，専用LSIもなく，比較的簡素な作りになっています。ここでは，SASIバスの扱いやSASIディスクへのコマンドについて説明します。

1 SASIバスの概要

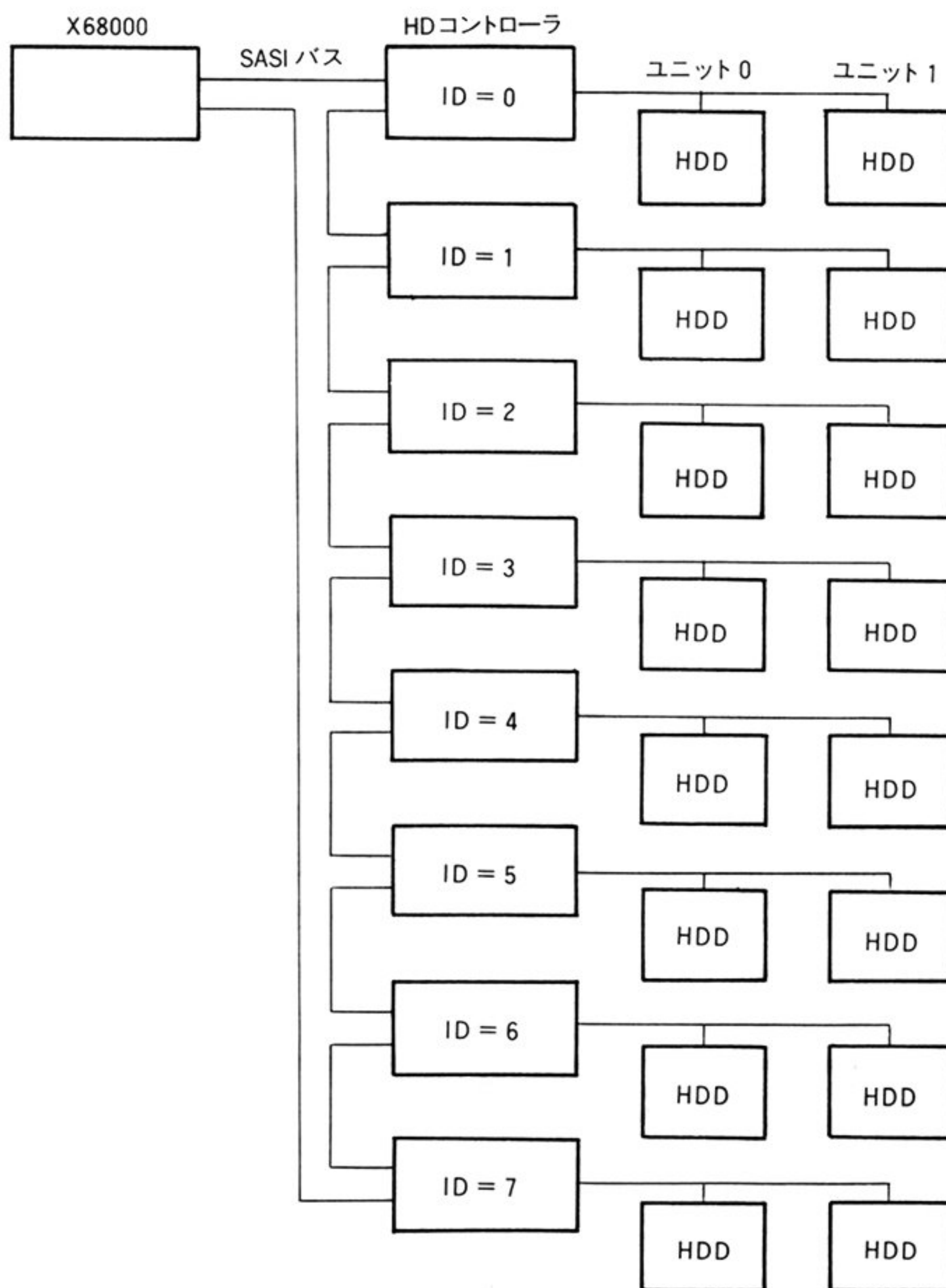
SASI (Shugart Associates System Interface) は，米国シュガート社が自社のハードディスクインタフェースとして設計したものです。パソコンの外付けハードディスク用のインタフェースとして普及し，利用されてきましたが，最近はSASIをもとにANSI (American National Standard Institute) で標準化が行われたSCSIバスに移行してきています。X 68000でも，外付けハードディスクインタフェースとしては初代機以来SASIインタフェースが装備されてきましたが，SUPER，XVIなどではSCSIに変更されています。

❶·1 SASIディスクの構成

SASIバスでは，バス上に最大8つのコントローラが接続できるようになっており，それぞれ固有の番号(ID)が振られています。また，SASIコマンドでは各コントローラのIDとは別に

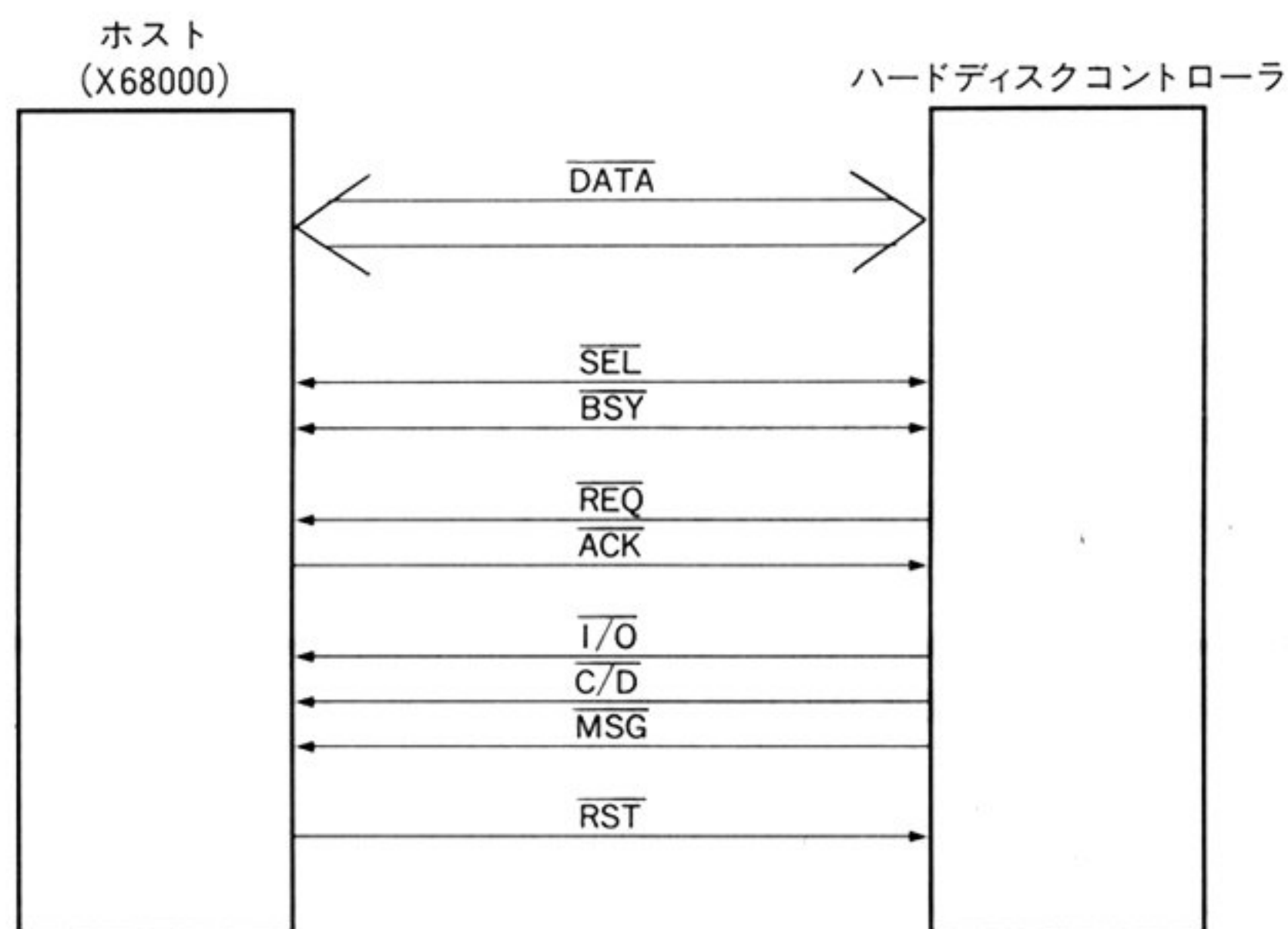
論理ユニット番号というものを設け，各コントローラの下に最大8台までのデバイスが接続できるようになっているため，理論上は最大64台までのデバイスが接続できることになります。ただし，市販されているほとんどのコントローラでは論理ユニット番号のうち'0'と'1'しかサポートされておらず，また，Human 68 Kでも，それに準じているため，実際に使用できるのは16台までとなっています。SASIハードディスクの接続例を図1に示します。

また，ハードディスク内蔵機では内蔵ディスクにIDを1つ使用してしまいますので，外部に接続できるのは14台，内部とあわせて15台までのディスクが使用可能となります。

●図……1　SASIハードディスクの接続

SASIバスによるアクセスでは，ホスト（この場合はX 68000）がデバイスをリード/ライトするのは一定の大きさのブロックと呼ばれる単位で行われ，ディスクの先頭から順番に振ったブロック番号によって，どのブロックをアクセスするかを指定します。ブロック番号からトラック番号やセクタ番号などへの変換はハードディスクコントローラ側で行いますので，ホストは実際のディスク上にどのようにデータが記録されているかを気にすることなく，単純にアクセスするブロック番号を指定するだけですむわけです。ブロックの大きさはX 68000用のディスクでは256バイトとなっています。

❶·2 SASIバス信号

SASIバスの信号を図2に示します。SASIバスは，8ビットのデータバスと8本の制御信号から構成されています。SASIバスの信号はどれも反転信号となっており，X 68000側で'1'をセットすると，バス上は'Low'レベルに，'0'だと'High'レベルになります。それぞれの信号の意味は次のようになっています。

●図……2　SASIバス信号

❶・❷1 DATA（データバス）

8ビットのDATAラインは，ホストとコントローラの間のコマンドやデータ，ステータスのやりとりなどを行うために使用されます。

❶・❷2 SEL（Select）

ホストが8台のコントローラの中からどれをアクセスするかを決めるために使用します。

❶・❷3 BSY（BUSY）

SASIバスが使用中であることを示す信号で，コントローラ側が出力します。ホストから選択されたコントローラはBSY信号を'1'('Low'レベル)にし，以後，コマンドの処理が完了するまで，BUSY状態を継続します。

❶・❷4 REQ（Request）

コントローラがホストにデータ転送を要求していることを示す信号です。通常は'0'('High'レベル)で，要求時'1'('Low'レベル)になります。ホストは，REQ信号に対してデータの読み出しや書き込みを行った後，ACK信号で応答します。

❶・❷5 ACK（Acknowlege）

REQ信号に対し，ホストが応答を示すために使用する信号です。リード要求だった場合にはDATAライン上のデータを引き取った後に，ライト要求だった場合にはDATAライン上にデータをセットした後にACKを'1'('Low'レベル)にし，コントローラがREQを'0'にしたらACKを'0'に戻すことで1回分のデータ転送が終了します。このようなやりとりの方法をREQ-ACKハンドシェークと呼ぶこともあります。

❶･❷6 I/O (Input/Output)

コントローラがホストに対して DATA ラインの方向（データの引き取りを要求しているのか，書き込みを要求しているのか）を示すために使用されます。'1'（'Low'レベル）のときにはコントローラからホスト（Input）方向，'0'のときにはホストからコントローラ（Output）方向であることを示します。

❶･❷7 C/D (Command/Data)

DATA ラインの内容がデータであるのか，コマンド/ステータスであるのかを示します。'0'（'High'レベル）のときにはデータであることを示します。

❶･❷8 MSG (Message)

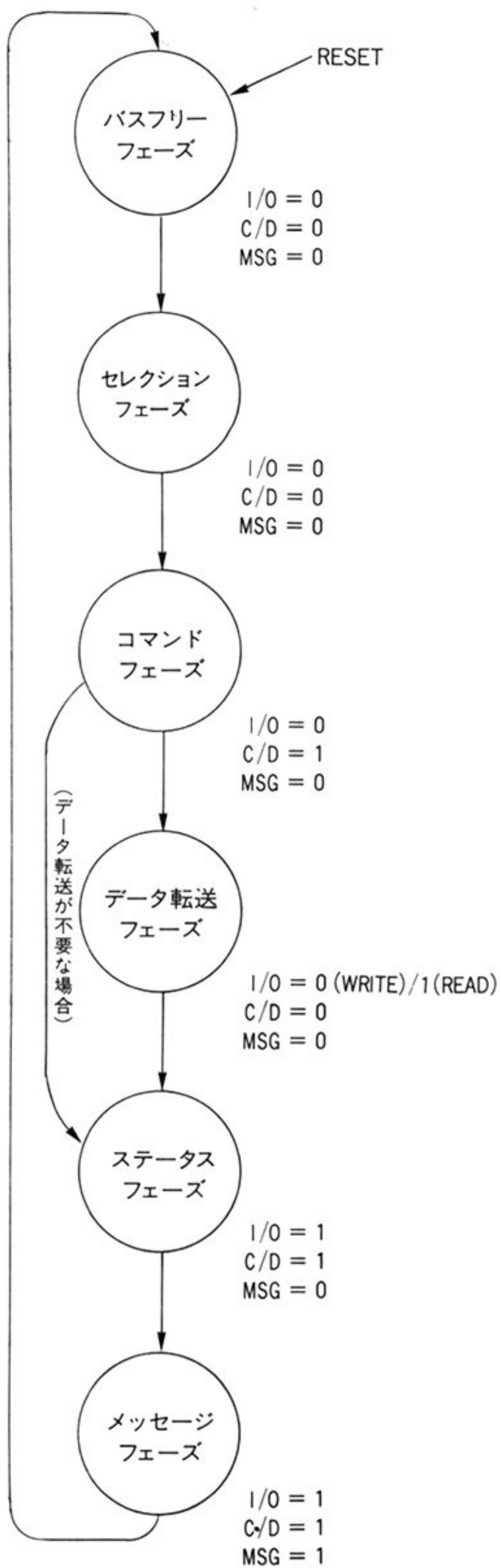
I/O, C/D ラインと組み合わされて，DATA ラインの内容がメッセージバイトであることを示します。メッセージバイトは，バス動作の最後に転送されるため，この信号が'1'('Low'レベル）のとき，バス動作の最後のサイクルであることを示すと考えることもできます。

❶･❷9 RST (Reset)

ホストが SASI バスを初期化するために使用する信号です。この信号を'1'（'Low'レベル）にすると，SASI バス上のコントローラはすべてリセットされます。データライト中であっても強制的にリセットされますので，使用にあたっては十分注意してください。

❶･3 SASIバスのフェーズ遷移

SASI バスはいくつかのバス状態を移行しながら動作します。この各状態をフェーズと呼びます。SASI の基本的なフェーズ遷移は 434 ページの図 3 のようになっています。

●図……3　SASIバス遷移図

❶・❸1 バスフリーフェーズ

バスが使用されていない状態であり，バス動作はここからスタートします。リセット後，バスはこの状態になります。

❶・❸2 セレクションフェーズ

ホスト（X 68000）が SASI 上の 8 つのコントローラの中からどれを使用するかを決めるフェーズです。

❶・❸3 コマンドフェーズ

セレクションフェーズで選択したコントローラに対して何を行うかを伝えるフェーズです。コマンドがディスクのリード/ライトやコントローラのステータス読み出しなど，データ転送を必要とするものであった場合にはデータ転送フェーズに，必要ない場合にはステータスフェーズに移行します。

❶・❸4 データ転送フェーズ

ホストとコントローラの間でデータの転送を行うフェーズです。転送するデータ量は，コマンドフェーズで与えたコマンドやパラメータで決まります。

❶・❸5 ステータスフェーズ

コントローラがホストに対してコマンドの実行結果を知らせるもので，1 バイトのデータが返されます。正常終了した場合は$00，なんらかのエラーがあった場合には$00 以外のデータ（通常，$02 を返すようです）が返されます。ホストは$00 以外が返されたら，REQUEST SENSE STATUS コマンドを使ってセンスステータスを引き取るようにします。

❶・❸6 メッセージフェーズ

転送サイクルの最後に行われるフェーズです。メッセージバイトと呼ばれる1バイトデータが返されます。一般的なSASIデバイスでは$00(コマンドコンプリートメッセージ)を返すだけのようです。

❶・4 SASIのバス動作

SASIバスのおおまかな動作について説明しましたので，次にSASIバスの動作を信号線の動きから見ていきましょう。図4にバスフリーフェーズから始まってふたたびバスフリーフェーズに戻るまでのSASIバスの動作の例を示します。この図では信号線が上にあるときが'1'(実際のバス上は'Low'レベル)，下にあるときが'0'（バス上は'High'レベル）となっています。また，これらの信号のうち，ホスト（X68000）側が操作するのはSELとACKのみで，残りはすべてコントローラ（ハードディスク）側が動かす信号です。データラインは省略しています。

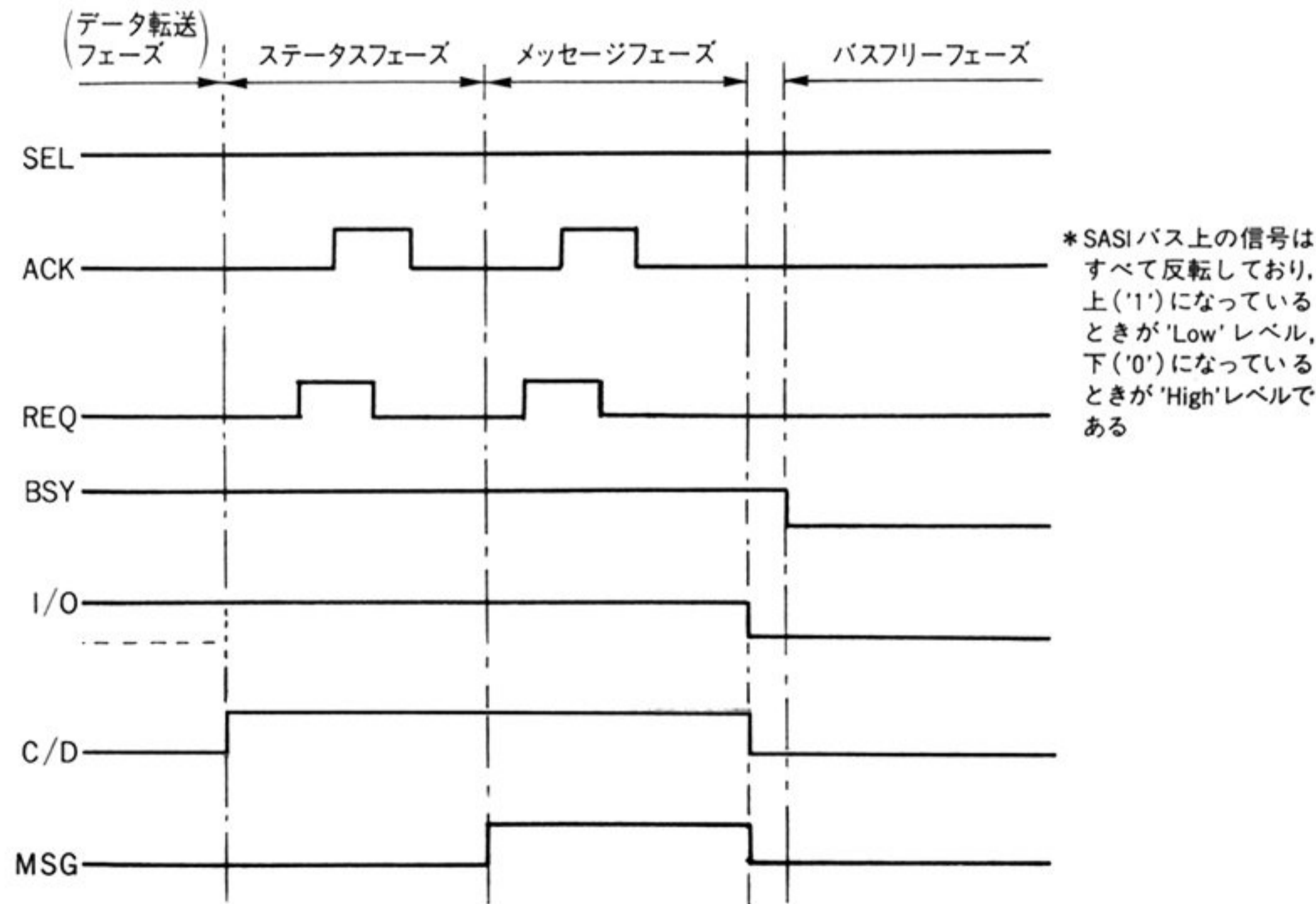
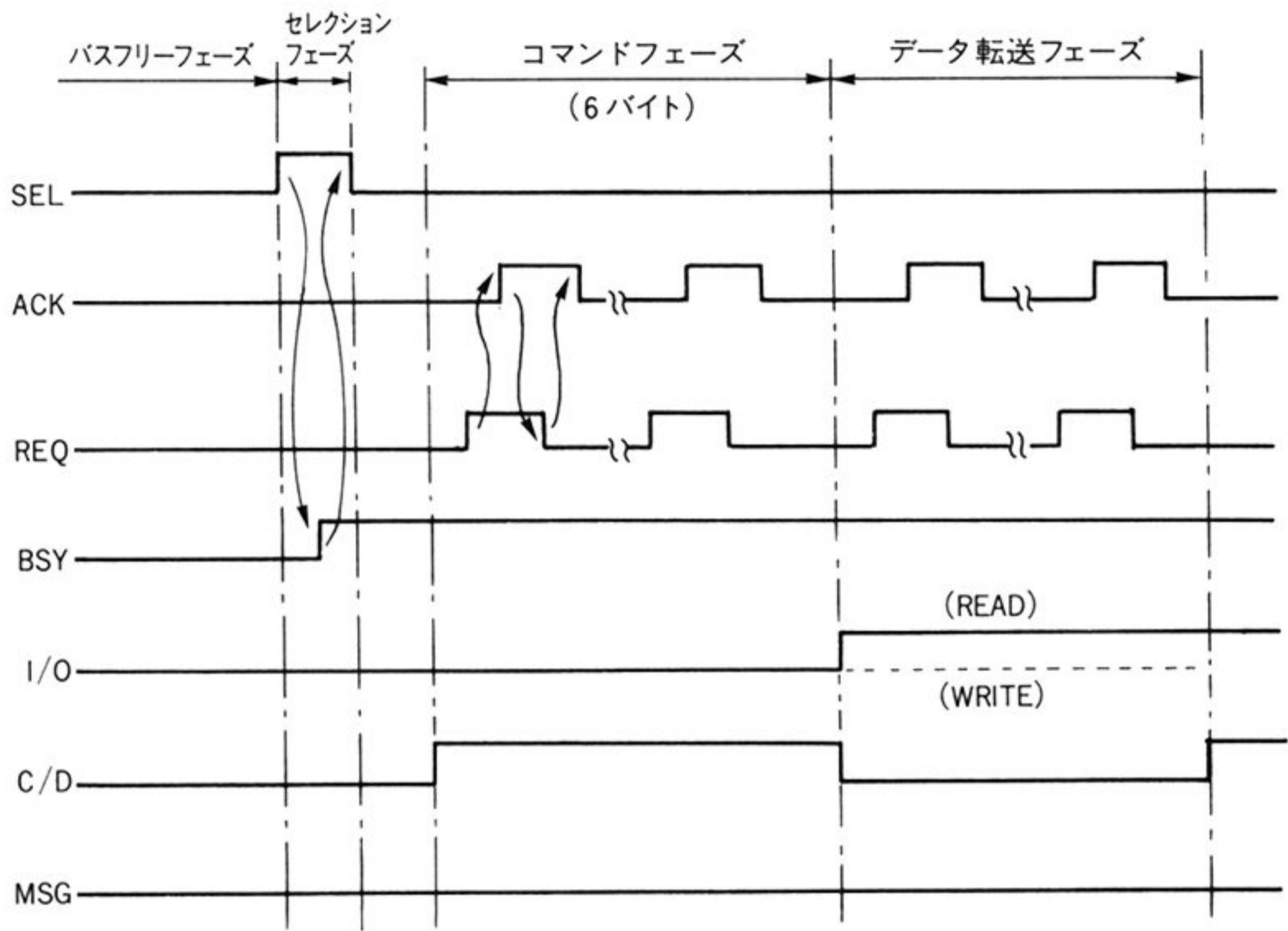
❶・❹1 バスフリーフェーズ

バスフリーフェーズ時は，すべての信号は'0'になっています。ホストは，バスがこの状態にあることを確認してからセレクションフェーズを開始します。

❶・❹2 セレクションフェーズ

ホストは，バス上にID番号をセットして，SEL信号を'1'にします。ID番号は，0～7がそれぞれデータラインのビット0～ビット7に対応しており，選択したいコントローラのID番号に対応するビットだけが'1'となったデータをバス上に出力します。たとえば，ID0のコントローラを選択するときは$01，ID3なら$08をバス上に出力させるわけです。

セレクションがうまくいくと，選択されたコントローラがBSY信号を'1'にして応答してきますので，SELを'0'に戻してセレクションフェーズを終了します。BSY信号は，最後のメッセージフェーズが終了するまで'1'になったままになります。これによって，SASIバスが現在

●図……4　SASIバス動作例

動作中であるか否かを判断することができます。

いつまでもBSYにならないときは，コントローラが存在しないものと見なして，SELを'0'に戻し，エラー終了させればよいでしょう。

X 68000ではセレクション用のポートがあり，そこにデータを書き込むと自動的にデータが出た後，SEL信号が動作するようになっています。

1·4 3 コマンドフェーズ

セレクションが終了すると，コントローラからコマンドの転送要求がきます。I/O，C/D，MSGは，それぞれ'0'，'1'，'0'（アウトプット方向，コマンド，メッセージではない）となり，REQ信号が'1'となってホストにコマンド転送を要求しますので，ホストはコマンドをデータラインにセットした後，ACKを'1'にしてコントローラに応答します。

コントローラは，ACKを受け取るとREQを'0'に戻しますので，ホストはこれをみてACK信号を'0'に戻します。

このようなREQ-ACKハンドシェークを繰り返して，コマンドとそれに付随するパラメータをコントローラに与えて，コマンドフェーズが終了します。SASIディスクの基本的なコマンドはすべて6バイト長ですので，ほとんどの場合，REQ-ACKハンドシェークは6回行われることになります。

X 68000では，データポートにアクセスすると，自動的にACK信号を返してくれるようになっていますので，ACK信号の操作を気にする必要はなく，REQ信号の監視だけをしていればよいようになっています。

1·4 4 データ転送フェーズ

データ転送フェーズでは，コントローラはC/Dを'0'に戻し，ホストからの書き込みの場合にはI/Oを'0'，読み出しの場合にはI/Oを'1'とします。MSGは'0'のまま保持されます。

信号線をこの状態に保ったまま，ふたたびREQ-ACKハンドシェークを行って必要な数のデータのやりとりが行われます。コマンドフェーズのときと同じようにデータポートへのアクセスで自動的にハンドシェークを行ってくれます。

データ転送はCPUで1つ1つ送るだけでなく，DMAで行うこともできます。後で紹介するサンプルプログラムでは，データ転送フェーズをDMA転送で行っていますので参考にしてください。

❶·❹5 ステータスフェーズ

ステータスフェーズは，C/D，I/Oとも'1'となり，1バイトのステータスバイトを送ってきます。ホストは，REQ-ACKハンドシェークでこのデータを引き取ります。

❶·❹6 メッセージフェーズ

コントローラ側はステータスフェーズが終了すると，MSG信号を'1'にしてメッセージフェーズであることを示し，メッセージバイトの引き取りを要求してきます。ホストは，REQ-ACKハンドシェークによって，このデータを引き取ります。これにより一連のバス動作が終了しますので，コントローラはBSYを含め，すべての信号を'0'にし，バスフリーフェーズに復帰します。

❶·5 SASIインタフェースポート一覧

SASIバスを制御するためのI/Oポートは図5のようになっています。

$E96001はSASIバスのデータのリード/ライトポートで，このポートにアクセスすると，自動的にREQ-ACKハンドシェークが行われます。これによりソフトウェアでREQ信号をチェックしながらACK信号を操作する手間を省くことができます。コマンドフェーズやデータ転送フェーズでは，DMAのチャンネル#1を使って転送を行うことができるようになっています。DMAリクエスト信号は，REQが'1'になったときにアクティブになり，ACK信号が'1'になったときに復帰します。

●図……5 SASIインタフェース ポートアドレス一覧

アドレス	READ/WRITE	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	備考
$E96001	R/W	DATA								SASIデータ入出力
$E96003	R	'0'			MSG	C/D	I/O	BSY	REQ	SASIステータス入力
	W	DATA								SEL信号を'0'(Hレベル)にする(DATAはSASIバスに出力される)
$E96005	W	データ任意								RST信号を約300μs間'1'(Lレベル)にする
$E96007	W	DATA								DATAをSASIバスに出力するとともにSEL信号を'1'(Lレベル)にする

$E96003番地は，読み出すとSASIの制御信号の状態が確認できます。CPUは，このポートを読み出すことで，現在のフェーズなどを知ることができます。

$E96003，$E96007番地の書き込みポートはセレクションフェーズ用につくられたポートで，データを書き込むと，そのデータがSASIバスに出力されるとともに，SEL信号が'0'ないし'1'にセットされます。$E96003番地への書き込みデータは通常$00にします。

$E96005番地への書き込みは，SASIバスのRESET信号を一定期間'1'にする信号です。このポートに書き込みを行うと，SASI上に接続されたハードディスクはすべてリセットされます。ディスクのリード/ライト中であっても強制的にリセットしてしまいますので，使用にあたっては十分注意してください。

❶·6 SASIのコマンド

SASIディスクで使用するコマンドは図6のような6バイトデータになっています。この6バイトデータを転送順序どおり上から順にコントローラに送るわけです。

先頭の1バイト目はディスク側が行うべき内容を示すもので，オペレーションコードと呼ばれます。オペレーションコードはさらに上位3ビットと下位5ビットに分けられ，上位3ビットで，そのコマンドが一般的な用途なのか，メーカ独自のコマンドなのかといったクラス分けを行うようになっています。通常使うコマンドはクラス0ですので，上位3ビットはすべて'0'になっています。

次のバイトの上位3ビットは論理ユニット番号です。3ビットありますので，1つのIDの下に最大8個のユニットまで接続できるわけですが，実際には論理ユニット番号として使われているのは0と1だけですので，上位2ビットはつねに'0'となります。

●図……6　SASIコマンドの一般形

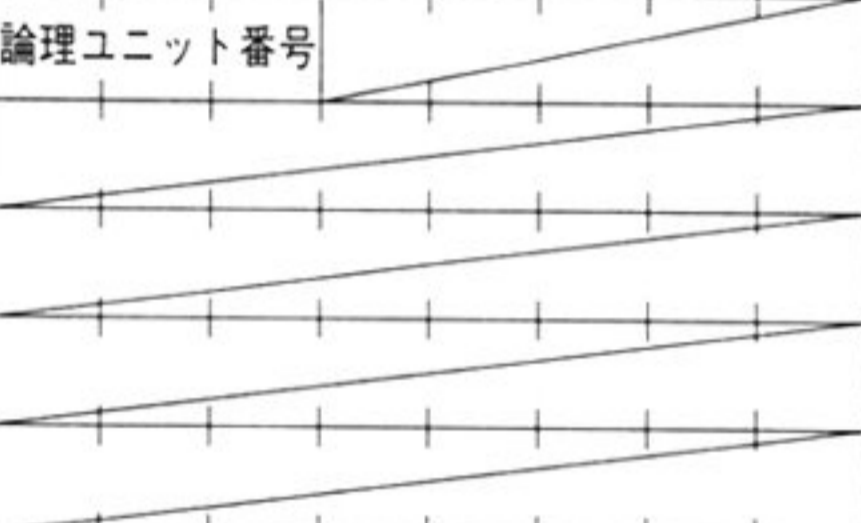
転送順序	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	備　考
0	コマンドクラス			コマンドコード					オペレーションコード
1	論理ユニット番号			論理アドレス(上位)					ユニット番号，アクセス開始ブロック
2	論理アドレス(中位)								
3	論理アドレス(下位)								
4	セクタブロック数								アクセスブロック数
5	コントロールバイト								通常すべて'0'で可

論理アドレスとセクタブロック数は，データをリード/ライトするときに有効なもので，リード/ライトを開始するブロック番号と，リード/ライトするブロックの数を指定します。1ブロックのサイズはX 68000では256バイトとなっています。

最後のコントロールバイトは，SCSIでは複数コマンドを連続実行するためのフラグなどに使われていますが，SASIではメーカによって扱いが異なるようですので，$00で使用するのが無難です。

❶·7 SASIの主要コマンド

SASIディスクのコマンドは，ディスクのリード/ライトなど，ごく基本的なもの以外に各メーカが独自に追加したものが数多くあり，「規格」とはとてもいえないような状況です。ここでは，これらの独自コマンドは無視し，どのメーカのものであってもほぼ備えていると思われる，主要な6つのコマンドについて説明しておくことにします。これらのコマンドの一覧を図7に示します。

なお，図8以降のコマンドの図の中で斜線が引いてあるビットは未使用の場合が多いのですが，メーカによっては勝手にメーカ独自の機能(たとえば，FORMAT DRIVEコマンドの論理アドレス部を，フォーマットを開始するブロック番号とするなど）に使用している場合がありますので，すべて'0'にしておくようにしてください。

●図……7　SASIの主要コマンド

コマンドの1バイト目			コマンド名	備　　考
オペレーションコード	コマンドクラス	コマンドコード		
$00	0	$0	TEST DRIVE READY	ドライブがレディ状態かチェックする
$01	0	$1	RECALIBRATE	ヘッドをトラック0に戻す
$03	0	$3	REQUEST SENSE STATUS	エラーステータスの引き取り
$04	0	$4	FORMAT DRIVE	ドライブの物理フォーマットを行う
$08	0	$8	READ	データの読み取り
$0A	0	$A	WRITE	データの書き込み

❶·❼1 TEST DRIVE READYコマンド

コマンドのフォーマットは図8のようになっています。このコマンドは，ディスクがレディ状態（動作可能な状態）にあるかどうかを調べるコマンドです。データ転送をともなわないので，コマンド送出後，ステータスフェーズに移ります。ディスクがレディならステータスフェーズで$00が返されます（レディ状態にないときに返される値はメーカによって異なります）。

●図……8 TEST DRIVE READY コマンド

転送順序	bit7	6	5	4	3	2	1	bit0	備　考
0	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	オペレーションコード：$00
1	論理ユニット番号								
2									
3									
4									
5									

❶·❼2 RECALIBRATEコマンド

ディスクのヘッドを0トラックに戻すコマンドです。コマンドのフォーマットは図9のようになっています。少し前のハードディスクではトラック0の位置だけはハード的にヘッド位置検出が行われますが，通常のトラック間移動の場合にはコントローラで前回との差分を判断して一定量移動させているだけでした。このため，一度ヘッドの位置がずれると，いくらヘッドを動かしてもずれたままとなり，エラーが頻発してしまいます。このようなときにはいったんハード的なセンサがあるトラック0に戻してからアクセスしなおすことで救われます。このために設けられたコマンドがRECALIBRATEコマンドというわけです。いまどきの小型のハードディスクはヘッドからの出力信号をみて自動的に微調整を行いますので，このようなコマンドにあまり意味はなくなりました。たんにヘッドを0トラックに移動させるために使用される程度でしょう。

●図……9　RECALIBRATE コマンド

転送順序	bit7	6	5	4	3	2	1	bit0	備　　考
0	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'1'	オペレーションコード：$01
1	論理ユニット番号								
2									
3									
4									
5									

❶・❼3　REQUEST SENSE STATUSコマンド

コマンドのフォーマットは図10のようになっています。エラーが発生した場合(ステータスフェーズのデータのビット1が'1'になっていたとき)，ホストは，このコマンドを送り，データ転送フェーズで4バイトのステータスを引き取ります。エラーが発生した後，このコマンドが発行されるかリセットされるまで，ステータスフェーズで渡されるデータは正常に戻らないのが普通です。センスバイトのフォーマットは444ページの図11のようになっています。先頭バイトで，エラーの内容や論理アドレスの内容が有効であるか否かが判断できるようになっているのですが，この内容は$00がエラーなしという以外はメーカごとに異なっています。

●図……10　REQUEST SENSE STATUS コマンド

転送順序	bit7	6	5	4	3	2	1	bit0	備　　考
0	'0'	'0'	'0'	'0'	'0'	'0'	'1'	'1'	オペレーションコード：$03
1	論理ユニット番号								
2									
3									
4									
5									

●図……11　センスバイトの構造

転送順序	bit 7	6	5	4	3	2	1	bit0	備　　考
0	V	エラークラス			エラーコード				V：論理アドレスの値が有効であることを示す('1'＝有効)
1	(自由に使用可)			論理アドレス(上位)					
2	論理アドレス(中位)								
3	論理アドレス(下位)								

❶・❼4　FORMAT DRIVEコマンド

ディスクを物理フォーマットするコマンドです。コマンドのフォーマットを図12に示します。コマンド発行後のフォーマット処理は，すべてコントローラ側で行ってくれますので，ホストはステータスフェーズに移るまで何もすることはありません。

●図……12　FORMAT DRIVEコマンド

転送順序	bit 7	6	5	4	3	2	1	bit 0	備　　考
0	'0'	'0'	'0'	'0'	'0'	'1'	'0'	'0'	オペレーションコード：$04
1	論理ユニット番号								
2									
3									
4									
5									

❶・❼5　READコマンド

ディスクの読み出しを行うコマンドです。読み出しを開始するブロック番号とブロック数を指定します。X 68000では1ブロックのサイズは256バイトです。コマンドのフォーマットは図13のようになっています。

●図……13 READ コマンド

転送順序	bit7	6	5	4	3	2	1	bit0	備　考
0	'0'	'0'	'0'	'0'	'1'	'0'	'0'	'0'	オペレーションコード：$08
1	論理ユニット番号			論理アドレス(上位)					読み出し開始ブロック番号
2	論理アドレス(中位)								
3	論理アドレス(下位)								
4	セクタブロック数								読み出すブロック数
5	'0'								

❶・❼6 WRITEコマンド

コマンドフォーマットは図14のようになっています。先頭ブロック番号とブロック数を指定して，ディスクへの書き込みを行います。コントローラは，最低でもディスクの1セクタ分のデータが揃うまでディスクへの書き込みは行いませんので，データ転送が遅くても問題はありません。

●図……14 WRITE コマンド

転送順序	bit 7	6	5	4	3	2	1	bit 0	備　考
0	'0'	'0'	'0'	'0'	'1'	'0'	'1'	'0'	オペレーションコード：$0A
1	論理ユニット番号			論理アドレス(上位)					書き込み開始ブロック番号
2	論理アドレス(中位)								
3	論理アドレス(下位)								
4	セクタブロック数								書き込むブロック数
5	'0'								

●2 サンプルプログラム

SASI ディスクの読み出しを行うサンプルプログラムをつくってみましたので参考にしてください。起動時のパラメータでブロック番号を指定すると，そのブロック(256バイト)の内容を表示します。エラー処理は何も行っていないので，ブロック番号が大きすぎたりすると，止まってしまいます。

このサンプルでは，データ転送フェーズをDMA転送で行っていますが，このときのDMAの転送モードは'11'(最初の1バイトだけがオートリクエスト，残りは外部転送要求)に設定しています。当初，たんなる外部転送要求でよいのではないかと思っていたのですが，実際に行ってみると，転送が途中で止まってしまうことが多かったのでモードを変更しました。この場合，最初の1バイト目はREQ信号の状態如何にかかわらず転送が行われてしまいますので，CPUでREQ信号が'1'になっているのを確認してから，DMAをスタートさせるようにしています。

●リスト……1　SASIディスクの読み出し

```
/*
 * ＳＡＳＩハードディスクアクセステスト
 *
 * XC ではvolatile がサポートされていないため、
 * 次の１行を入れてvolatileを無効にしてください
 * #define  volatile
 */

#include <doslib.h>

struct  DMAREG {
    unsigned char   csr;
    unsigned char   cer;
    unsigned short  spare1;
    unsigned char   dcr;
    unsigned char   ocr;
    unsigned char   scr;
    unsigned char   ccr;
    unsigned short  spare2;
```

```
    unsigned short  mtc;
    unsigned char   *mar;
    unsigned long   spare3;
    unsigned char   *dar;
    unsigned short  spare4;
    unsigned short  btc;
    unsigned char   *bar;
    unsigned long   spare5;
    unsigned char   spare6;
    unsigned char   niv;
    unsigned char   spare7;
    unsigned char   eiv;
    unsigned char   spare8;
    unsigned char   mfc;
    unsigned short  spare9;
    unsigned char   spare10;
    unsigned char   cpr;
    unsigned short  spare11;
    unsigned char   spare12;
    unsigned char   dfc;
    unsigned long   spare13;
    unsigned short  spare14;
    unsigned char   spare15;
    unsigned char   bfc;
    unsigned long   spare16;
    unsigned char   spare17;
    unsigned char   gcr;
} ;

volatile struct DMAREG  *dma;
volatile unsigned char  *sasi_data;
volatile unsigned char  *sasi_status;
volatile unsigned char  *sasi_sel_off;
volatile unsigned char  *sasi_reset;
volatile unsigned char  *sasi_sel_on;

#define BUFSIZE 0x100
unsigned char   diskbuf[BUFSIZE];

#define BUSFREE_PHASE       0x00
#define SELECTION_PHASE     0x02
```

```
#define COMMAND_PHASE       0x0a
#define DATA_READ_PHASE     0x06
#define STATUS_PHASE        0x0e
#define MESSAGE_PHASE       0x1e

#define REQ_BIT         0x01


void main();
void sasi_select();
void sasi_send_command();
void sasi_send_a_byte();
unsigned int sasi_get_status();
unsigned int sasi_get_message();
void wait_sasi_status();
void dma_setup();
void dma_start();
void dma_stop();
void wait_complete();
void clear_flag();

void main(argc,argv)
    int argc;
    char    *argv[];
{
    unsigned int    i,j,id,blk_no,blk_h,blk_m,blk_l;
    unsigned char   c;
    if (argc >= 2)
        blk_no = atoi(argv[1]);
    else    blk_no = 0;
    if (argc >= 3)
        id = atoi(argv[2]);
    else    id = 0;
    blk_l = blk_no & 0xff;
    blk_m = (blk_no >> 8) & 0xff;
    blk_h = (blk_no >> 16) & 0x1f;
    printf("Block# = %d(%06X)[%02X:%02X:%02X] Drive = %d¥n",
                blk_no, blk_no, blk_h, blk_m, blk_l, id);
    for (i=0; i<BUFSIZE; i++)
        diskbuf[i] = 0;
```

```
    SUPER(0);
    dma             = (struct DMAREG *)0xe84040;
    sasi_data       = (unsigned char *)0xe96001;
    sasi_status     = (unsigned char *)0xe96003;
    sasi_sel_off    = (unsigned char *)0xe96003;
    sasi_reset      = (unsigned char *)0xe96005;
    sasi_sel_on     = (unsigned char *)0xe96007;
    clear_flag();
    dma_setup();
    sasi_select(id);
    sasi_send_command(8,blk_h,blk_m,blk_l,1,0);
    wait_sasi_status(DATA_READ_PHASE | REQ_BIT);
    dma_start();
    wait_complete();
    clear_flag();
    printf("STATUS = ");
    printf("%02X¥n",sasi_get_status());
    printf("MESSAGE = ");
    printf("%02X¥n",sasi_get_message());
    for (i=0; i<BUFSIZE; i+=0x10) {
        for (j=0; j<0x10; j++)
            printf("%02X ",diskbuf[i+j]);
        for (j=0; j<0x10; j++) {
            c = diskbuf[i+j];
            if ((c < 0x20) ||(c >= 0xe0)|| ((c >= 0x80) && (c < 0xa0)))
                printf(".");
            else    printf("%c",diskbuf[i+j]);
        }
        printf("¥n");
    }
}

void sasi_select(id)
    unsigned int    id;
{
    unsigned int    stat;
    if (stat = *sasi_status) {
        printf("SASI stat = %d¥n",stat);
        exit(1);
    }
    *sasi_sel_on = 1 << id;
```

```
    wait_sasi_status(SELECTION_PHASE);
    *sasi_sel_off = 0;
}

void sasi_send_command(p1,p2,p3,p4,p5,p6)
    unsigned int    p1,p2,p3,p4,p5,p6;
{
    sasi_send_a_byte(p1);
    sasi_send_a_byte(p2);
    sasi_send_a_byte(p3);
    sasi_send_a_byte(p4);
    sasi_send_a_byte(p5);
    sasi_send_a_byte(p6);
}

void sasi_send_a_byte(dat)
    unsigned int    dat;
{
    wait_sasi_status(COMMAND_PHASE | REQ_BIT);
    *sasi_data = dat;
}

unsigned int sasi_get_status()
{
    wait_sasi_status(STATUS_PHASE | REQ_BIT);
    return((unsigned int)*sasi_data);
}

unsigned int sasi_get_message()
{
    wait_sasi_status(MESSAGE_PHASE | REQ_BIT);
    return((unsigned int)*sasi_data);
}

void wait_sasi_status(dat)
    unsigned int    dat;
{
    while(*sasi_status != dat)
        ;
}
```

```
void dma_setup()
{
    dma->dcr = 0x80;
    dma->ocr = 0xb3;
    dma->scr = 0x04;
    dma->ccr = 0x00;
    dma->cpr = 0x08;
    dma->mfc = 0x05;
    dma->dfc = 0x05;
    dma->mtc = BUFSIZE;
    dma->mar = diskbuf;
    dma->dar = (unsigned char *)sasi_data;
}

void dma_start()
{
    dma->ccr |= 0x80;
}

void wait_complete()
{
    while(!(dma->csr & 0x90))
        ;
}

void clear_flag()
{
    dma->csr = 0xff;
}
```

SCSI

SCSIインタフェースは，SUPER以降内蔵され，CD-ROMなど新しいデバイスへの対応も期待されています。ここでは，SCSIコントローラLSIの扱い方と，SCSIディスクのコマンドについて説明します。

1 SCSIの概要

SCSIインタフェースは，SASIをもとに複数ホストへの対応，コマンドの機能拡張などを行い，ANSIで規格化したものです。SCSIとSASIと比べたときのおもな違いをあげると，次のようになります。

- 複数ホストの構成に対応した
- 時間のかかるコマンド処理の場合にいったんバスを切り離し(ディスコネクト)，後で再接続する（リコネクト）機能が追加された
- バスの使用権の調停をするアービトレーションフェーズ，再接続のためのリセレクションフェーズが追加された
- イニシエータからターゲットへのメッセージ転送機能が追加された
- メッセージフェーズ，ステータスフェーズで返される値が規格化された
- 複数コマンドの連続実行を指定するコマンドリンク機能の追加が行われた
- リード/ライトコマンドやセンスデータに拡張フォーマットが定義された

これらの変更にともない，呼び方もいくつか変更されています。大きな変更点としては，SASIのホストとコントローラという名称がイニシエータとターゲットという名称となったこと，メッセージ転送が双方向になったことから，イニシエータからターゲットへのメッセージ転送フェーズをメッセージアウトフェーズと呼ぶようになり，SASIのメッセージフェーズはメッセージインフェーズと改名されたという2点があげられるでしょう。

❶·1 SCSIバスの構成

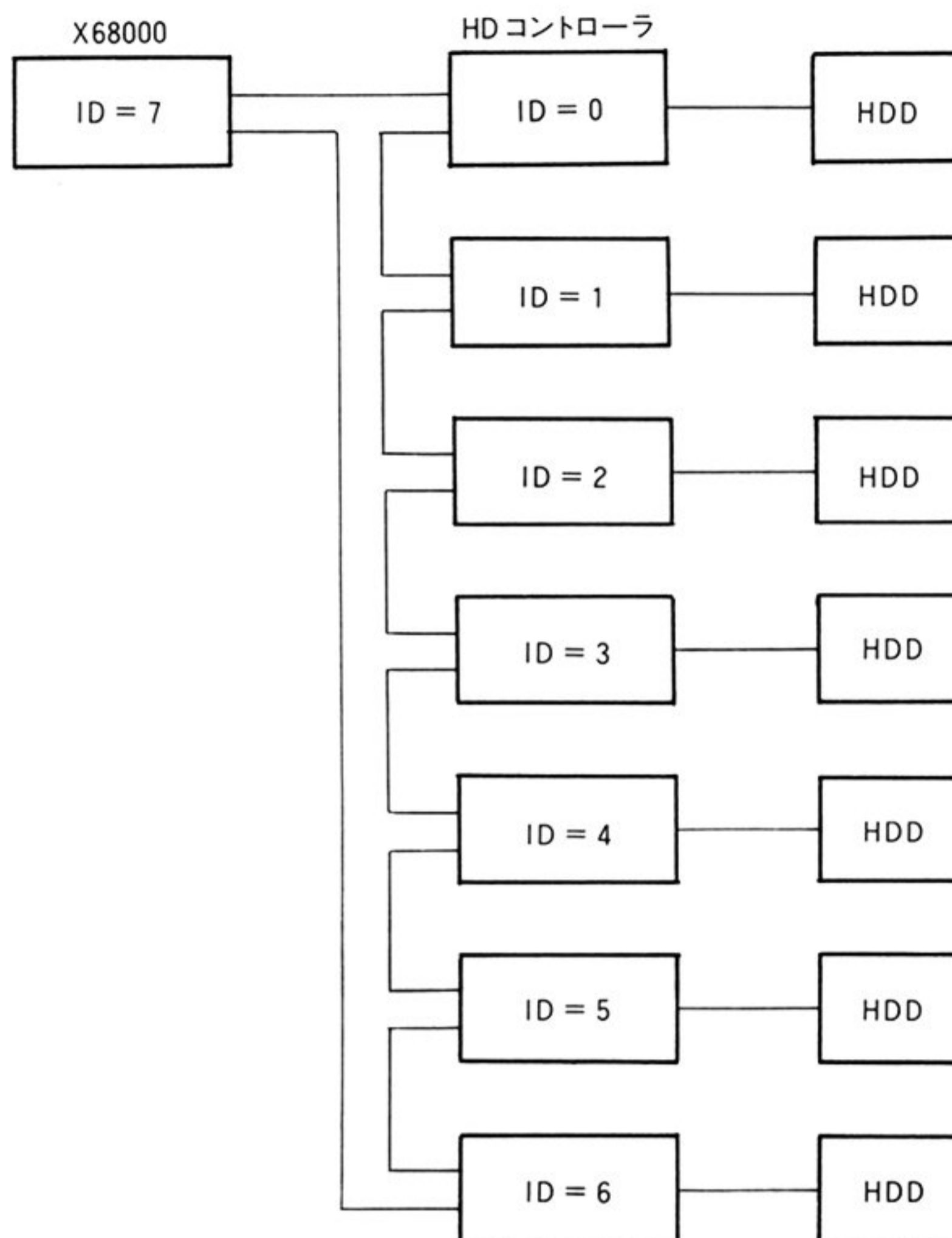
SCSIバスは，SASIと同様に最大8個のデバイスを接続できるようになっています。ただし，SCSIの場合にはイニシエータ（X 68000本体）自体もIDを必要としますので，バス上に接続できるのは7個までとなります。SCSI上の0から7までのIDのうち，X 68000はデフォルトではID#7を使用しています。

SCSIも，SASI同様に，論理ユニット番号を使用することで各IDの下に8台のユニットが，さらにSCSIの拡張仕様として設けられたEXTENDED　IDENTIFYメッセージを利用すると，各IDの下に2048台のユニットが接続できることになるのですが，シャープが提供している標準のSCSIドライバでは，これらをまったく使用していない（コマンド中の論理ユニット番号は0のみとなります）ため，SCSIバス上に接続できるディスクは最大7台となっています。SCSIバスへのディスクの接続例を図1に示します。

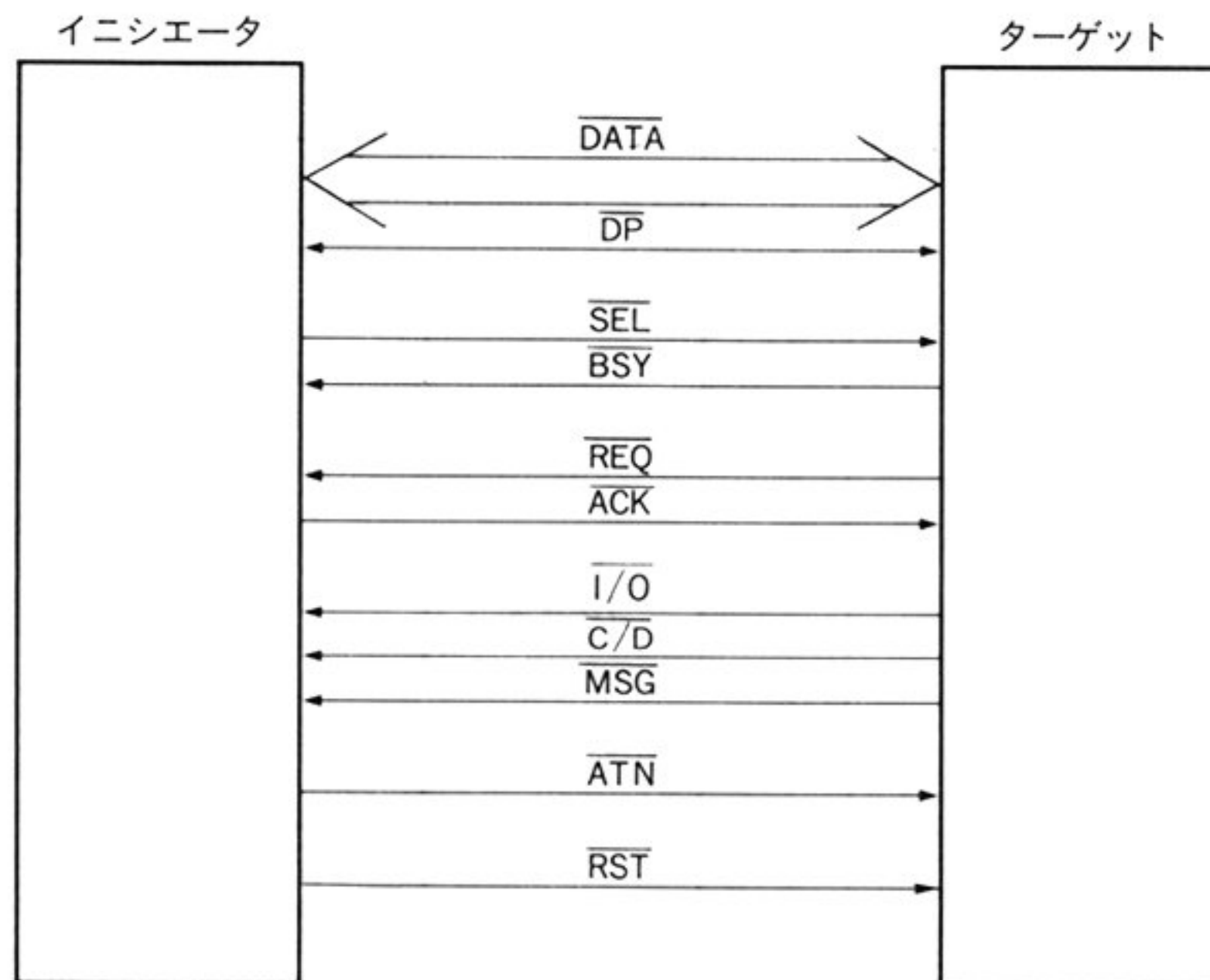
アクセスの単位であるブロックの大きさは，Human 68 Kの場合，SASIでは256バイト固定でしたが，SCSIドライバでは256バイト，512バイト，1024バイトのいずれでもかまわないようになっています。

❶·2 SCSIバス信号

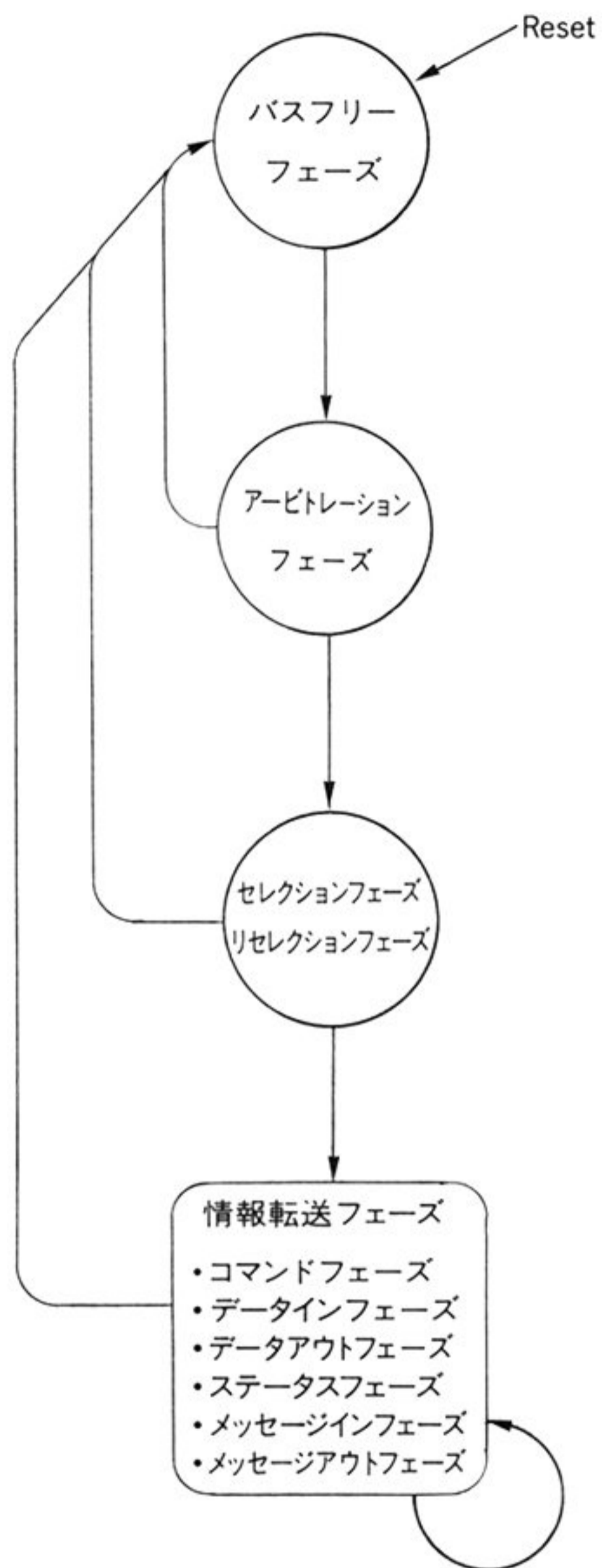
SCSIバスの信号を456ページの図2に示します。信号線としてはSASIが持っていた信号に，ATN信号とDP（パリティ）が追加されたものとなっています。SCSI対応デバイスの傾向としてはパリティを使用するものが多くなってきていますが，X 68000のSCSIドライバは，SCSIコントローラをパリティディセーブル（自分がデータを出力するときにはパリティを出力しますが，データ入力のときのパリティチェックは行わないモード）にプログラムして使っていますので，接続されるデバイスはパリティを使用していなくてもかまいません。

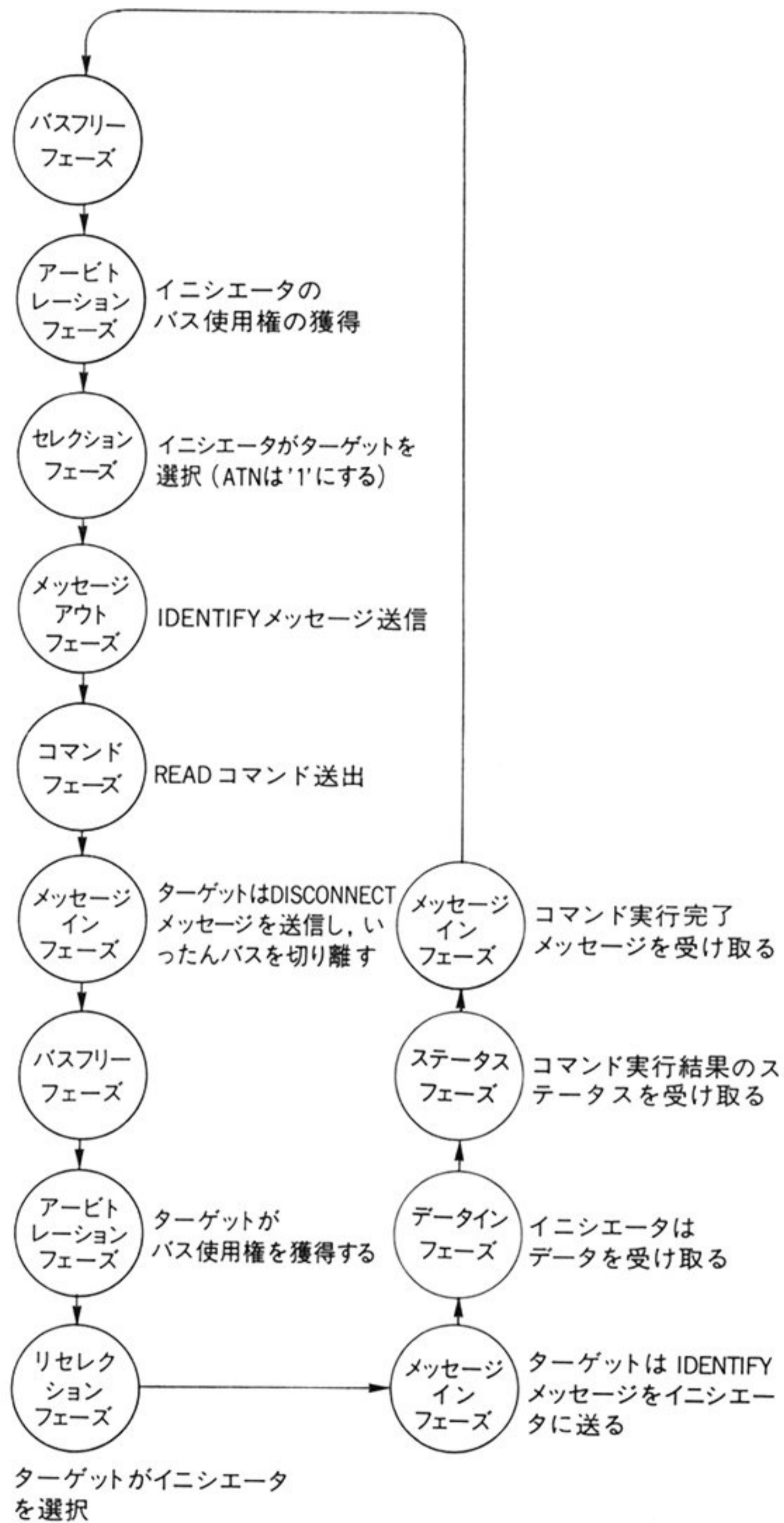
●図……1　SCSIディスクの接続

❶·❷1　ATN信号

SCSIになって追加されたATN信号は，イニシエータ（通常はX 68000）からターゲット（ディスクなど）に対して，データ転送中のエラー通知や動作モード設定などを要求する信号です。イニシエータとターゲットがはっきりしている状態（バスフリーフェーズやバス使用権の調停を行うアービトレーションフェーズでない状態）であれば，イニシエータは，いつでもATN信号を'1'（Lowレベル）にしてターゲットに通知したい内容があることを示すことができます。通知する内容を実際に送るフェーズは，メッセージアウトフェーズと呼ばれます。ターゲットは都合のよいときにメッセージアウトフェーズに移行して，イニシエータからの通知を受け取ります。

●図……2　SCSIバス信号

❶·3　SCSIバスのフェーズ遷移

SASIからSCSIへの移行で増やされたフェーズは，バス使用権の調停をするアービトレーションフェーズ，ATN信号のところでも触れたメッセージアウトフェーズ，ターゲットからイニシエータへ再接続を要求するリセレクションフェーズの3つだけなのですが，これらのどれもがフェーズ遷移として整理しにくいものであることから，一般的な形に表そうとすると図3のようになります。

●図……3　SCSIバス遷移図（規格書より）

これではいささかわかりにくいので，具体的なフェーズ遷移の例をディスクの読み出しを例に図示してみたのが458ページの図4です。これをもとに，バス遷移をかんたんに説明しておきましょう。この例では，コマンドを受け取った後，いったんバスを切り離し，読み出すデータが揃った時点で再接続するという，ややSCSIらしい動作を行わせています。

●図……4　SCSIバス動作例（ディスクRead）

❶·❸1 バスフリーフェーズ

バスがまったく使用されていない状態です。バス動作はここからスタートします。

❶·❸2 アービトレーションフェーズ

バスの使用権の調停を行います。バスを使用したいものが，データバス上に各自の ID 番号を出力し，もっとも優先度の高い（もっとも優先度が高いのは ID＃7：通常は X 68000 本体）ものがバスの使用権を獲得します。ここで負けたものはふたたびバスフリーフェーズになるまで待たされます。

❶·❸3 セレクションフェーズ

バスの使用権が得られたイニシエータは，セレクションフェーズによってターゲットを選択します。SASI のセレクションフェーズと同じようなものですが，データバス上にはターゲットの ID だけでなく，自分の ID 番号にあたるビットも'1'にするところが違います。これは，ターゲットに対して誰が自分にアクセスしにきたかを伝え，後で述べるリセレクションフェーズが実行できるようにするためです。

また，この例ではセレクションフェーズのときに ATN を'1'にしています。ATN を'1'にするのはオプションであり，'0'のままにしておいてもかまわないのですが，この例では，ターゲットに対してディスコネクト処理を行ってもよいことを伝えたいので，ATN 信号を使用し，次のメッセージアウトフェーズを要求しています。

❶·❸4 メッセージアウトフェーズ

セレクションフェーズのときに ATN を'1'として選択されたので，ターゲットはコマンドフェーズに移る前にメッセージアウトフェーズに移行してイニシエータからのメッセージを受け取ります。

イニシエータは，ここで IDENTIFY メッセージを送り，この中でディスコネクト処理有効を伝えます。

❶·❸5 コマンドフェーズ

イニシエータからのメッセージを受け取ったターゲットは，次にコマンドフェーズに移行し，イニシエータからのコマンドを受け取ります。このフェーズの動作はSASIのときとなんら変わりません。

❶·❸6 メッセージインフェーズ

SASIでは，この後，実際のデータ転送が始まるまでバスはBUSYになったまま（BSY信号が'1'になったまま）でしたが，実際にはREADコマンドを受け取った後，データが揃うまでにはかなり時間がかかるため，ターゲットはここでバスをいったん切り離し，バスフリーフェーズに移行します。

ターゲットは，メッセージインフェーズ（SASIでいう，メッセージフェーズ）に移行し，イニシエータに対してDISCONNECTメッセージを送り，バスの一時切り離しを通知します。この後，イニシエータ，ターゲットともバスの使用権を放棄し，SCSIバスはふたたびバスフリーフェーズに移行します。

❶·❸7 バスフリーフェーズ

いちばん最初のバスフリーフェーズとなんら変わるところはありません。ただ，内部的には，先ほどまでのイニシエータはターゲットからのリコネクトを待っていますし，ターゲットはデータの読み出しを行い，イニシエータへの転送の準備を行っています。

❶·❸8 アービトレーションフェーズ

データの用意ができたターゲットは，アービトレーションフェーズに参加し，バスの使用権獲得を行います。ここで負ければ，次のバスフリーフェーズまで待たされることになります。

❶・❸ 9 リセレクションフェーズ

バスの使用権を獲得できたターゲットは，リセレクションフェーズに入り，イニシエータと再接続します（ここでリセレクションでなく，セレクションフェーズに入ってしまうと，自分がイニシエータとして動作することになってしまいます)。最初のセレクションフェーズでイニシエータが渡したIDは，このフェーズで必要になるわけです。

❶・❸ 10 メッセージインフェーズ

ターゲットはイニシエータに対してIDENTIFYメッセージを送ります。このメッセージの中にある論理ユニット番号によって，イニシエータはどの論理ユニットの処理結果を受け取るのかを知ることができます。

❶・❸ 11 データインフェーズ

ターゲットからイニシエータに対し，ディスクから読み出したデータの転送を行います。転送はSASI同様，REQ-ACKハンドシェークで行われます。

❶・❸ 12 ステータスフェーズ

SASIのときと同様，コマンド実行結果のステータスを受け取ります。SASIでは$00の正常ステータス以外はメーカごとに勝手に割り振っていましたが，SCSIではステータス番号と内容が規定されています。

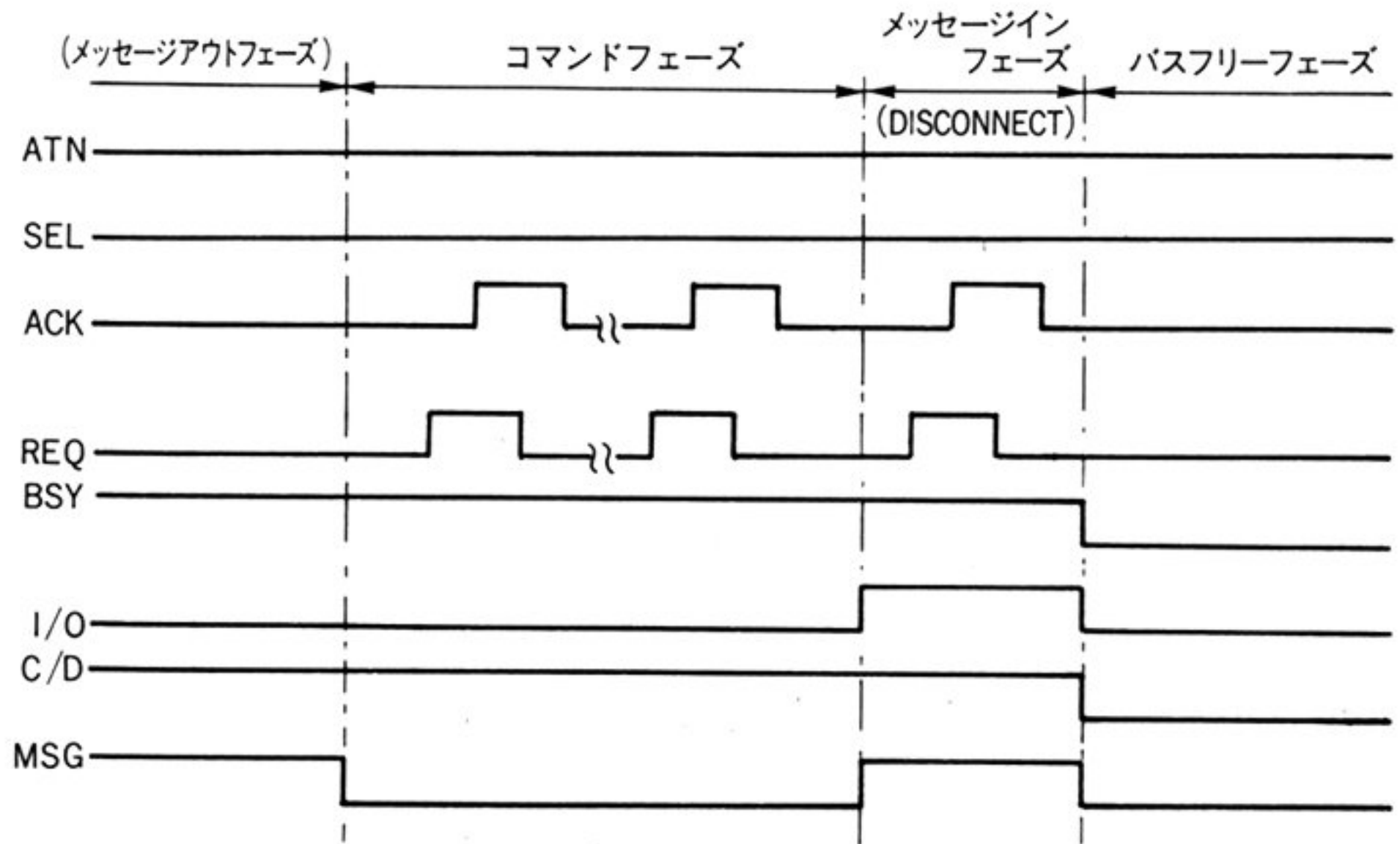
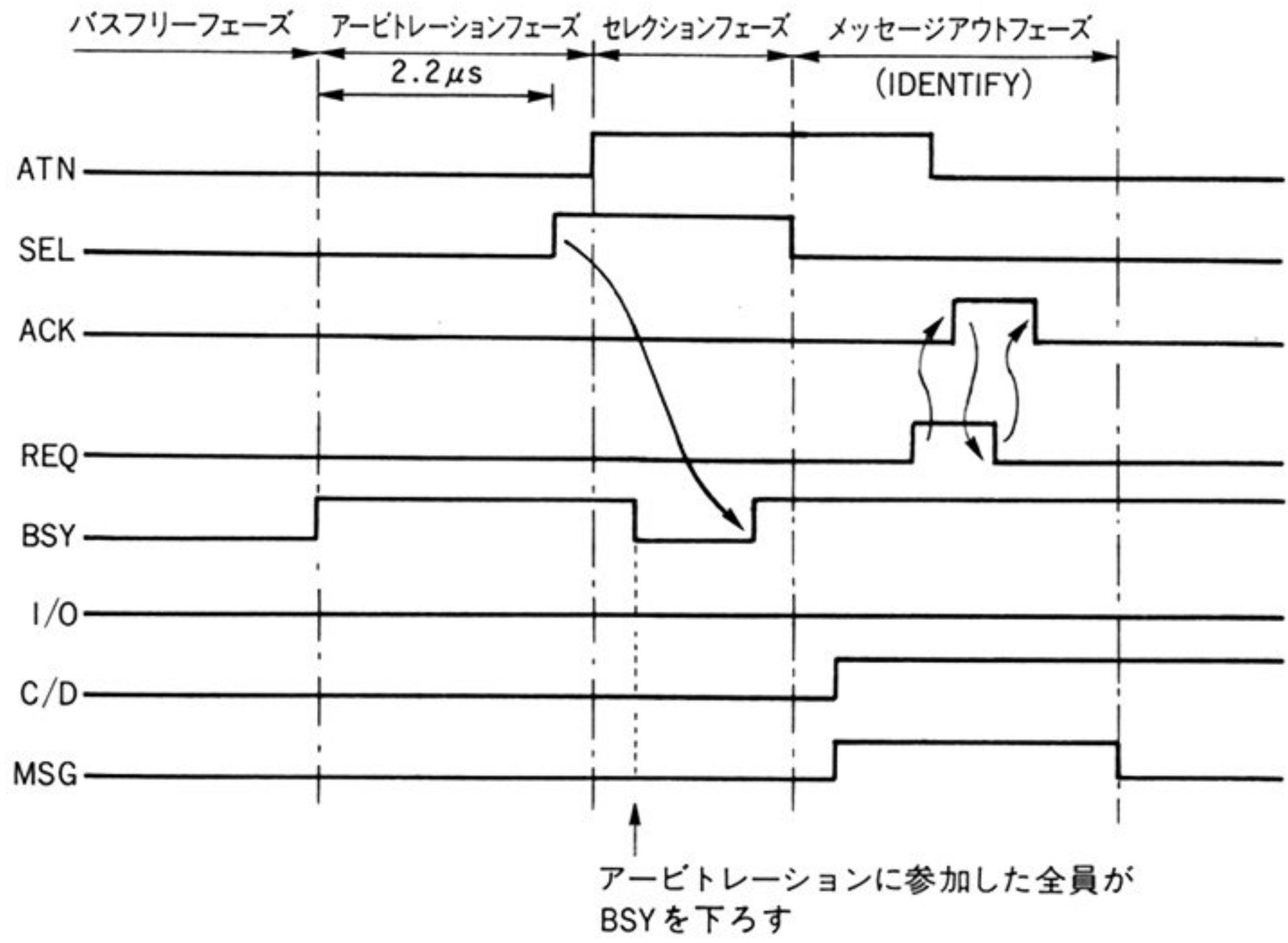
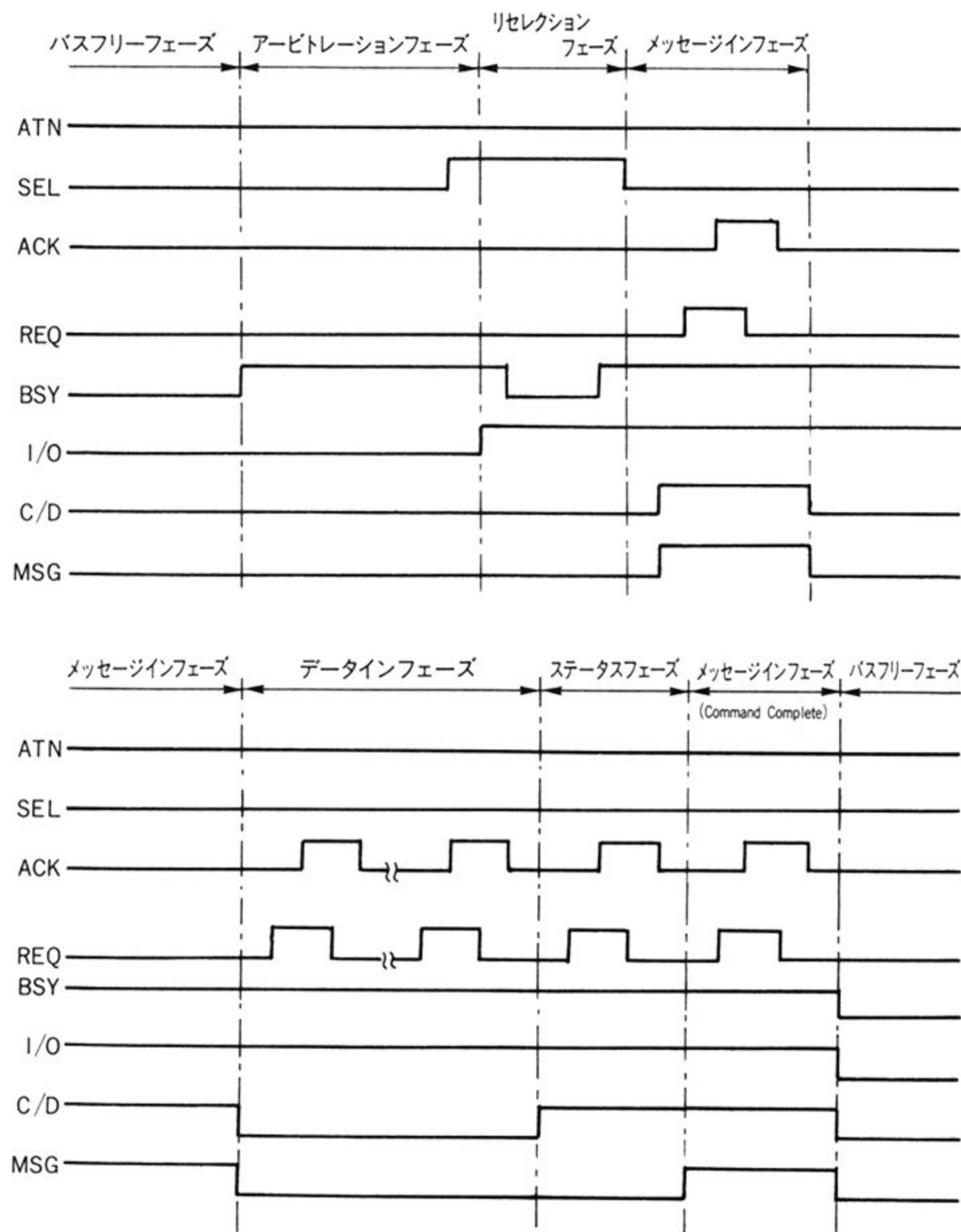
❶・❸ 13 メッセージインフェーズ

SASIのときのメッセージフェーズと同様です。通常は$00 (COMMAND COMPLETE)メッセージが返されます。これでコマンド処理は終了し，SCSIバスはふたたびバスフリーフェーズになります。

SASIのときに比べると，面倒になったように思えますが，これはディスコネクト/リコネクト機能を使っているためです。SCSIでは，これらを使用しない動作も可能となっており，この場合の動作はセレクションの前にアービトレーションフェーズがくる以外はSASIのときとほとんど同じです。後に紹介するサンプルプログラムでも，簡略化のため，ディスコネクト/リコネクト機能は使用していません。

❶·4 SCSIのバス動作

先ほどのフェーズ遷移を信号の動きで追いかけたのが図5と図6です。ほとんどはSASIと同じですので，ここでは追加されたアービトレーション，メッセージアウト，リセレクションの各フェーズを見ておくことにしましょう。

●図……5　SCSIバス動作例（その1）

●図……6　SCSIバス動作例（その2）

❶·❹1 アービトレーションフェーズ

アービトレーションフェーズは，データライン上の自分のIDに相当するビットとBSY信号を'1'（'Low'）にすることで開始されます。アービトレーションに参加したいデバイスは，BSY信号が'1'になってから，1.8μs以内に自分のIDに相当するビットを'1'にします。

BSYが'1'になってから2.2μs後に，データラインが読み出されます。自分のIDよりも優

先順位の高いビットが'1'になっていないときは，そのデバイスがバスの使用権を獲得します。優先順位は固定で，ID＃7がもっとも高く，ID＃0がもっとも低くなっています。X 68000のSCSIドライバでは，自分のIDのデフォルト値を＃7に設定しています。

❶・❹ 2 メッセージアウトフェーズ

I/O, C/D, MSGがそれぞれ'0', '1', '1'となります。メッセージインフェーズのときとはI/Oが逆になっています。メッセージデータのやりとりをREQ-ACKハンドシェークで行うのはSASIのときと変わりません。

❶・❹ 3 リセレクションフェーズ

セレクションフェーズではI/O, C/D, MSGがそれぞれ'0', '0', '0'でしたが，リセレクションフェーズではデータバスの向きがターゲットからイニシエータに向きますから，I/Oが逆になり，'1', '0', '0'の状態でSEL信号が'1'になります。

●2 X68000のSCSIインタフェースの概要

X 68000のSCSIインタフェースは，オプションボードで対応するものと標準で内蔵したものの2種類があります。これらは使っているLSI（SPC：SCSIプロトコルコントローラ）こそ同じですが，ポートアドレスや割り込みなどは変更されており，SCSI内蔵モデルにSCSIインタフェースボードを取り付けることも可能になっています。

また，これらとあわせ，SCSI対応にするため，従来未使用であったSRAMの領域に新たな情報の追加などが行われています。

ここでは，これらの機種間の違いや，新たに追加された情報などについて説明します。

❷·1 SCSI関連ポート, 割り込み

X 68000のSCSIインタフェースのポートアドレスや割り込みの配置などは図7のようになっています。表中, SCSI-ROMというのは, SCSIからブートするためのIPLなどが書き込まれたROM, SCSI-ROM識別ラベルは, そのアドレスにあるものがSCSI-ROMであることを識別するために書き込まれている文字列です。SCSI内蔵タイプでは, $FC0024からの5バイトに'SCSIIN'という文字列が, CZ-6BS1では$EA0044からの5バイトに'SCSIEX'という文字列が書き込まれています。

●図……7 SCSI関連アドレス等

条 件	SCSI内蔵モデル	拡張ボード(CZ-6BS1)
SPCのポートアドレス	$E96021～$E9603F	$EA0001～$EA001F
SCSI-ROMのアドレス	$FC0000～$FC1FFF	$EA0020～$EA1FFF
SPCの割り込みレベル	レベル1	レベル2と4を選択可
〃 ベクタ	$6C	$F6
SCSI-ROM識別ラベル	$FC0024～$FC0029 $53435349494E (SCSI IN)	$EA0044～$EA0049 $534353494558 (SCSIEX)

❷·2 IPL-ROMの内容

X 68000のSCSI内蔵でないモデルでは, IPL-ROM領域256Kバイトのうち, 前半の$FC0000～$FDFFFFの128Kバイトの領域は使用されておらず, アクセスすると, 後半($FE0000～$FFFFFF)と同じものが読み出されるようになっていましたが, SCSI内蔵モデルでは, この領域のうち, $FC0000～$FC1FFFの8KバイトをSCSI用のIPLプログラム領域として使用し, 残りの$FC2000～$FDFFFFはすべて$FFになっています。

また, SCSI内蔵モデルでは, 内部メモリ容量のデフォルト値を2Mバイトとしているため, IPL-ROM中の$FF079B番地の内容が$10から$20に変更されています。

❷·3 SRAMの内容

SCSI対応化にともない, SRAMの$ED006F, $ED0070, $ED0071番地が使用されるようになりました。この内容を図8に示します。

●図……8　SRAMの追加情報

$ED006F番地は，$ED0070, $ED0071の内容が有効であるか否かのフラグとして用いられており，有効であるときは$56(ASCIIコードで'V')が書き込まれます。'V'が書き込まれていない場合には，SCSIローダプログラムが'V'を書き込むとともに，$ED0070を$07, $ED0071を$00に設定します。

$ED0070のビット3は，SCSI内蔵タイプにSCSIオプションボードを取り付けた場合，どちらのSCSIを使用するかのフラグで，'0'のときは内蔵SCSI，'1'のときにはSCSIオプションボードを使用します。$ED0070の下位3ビットは自分自身のID番号です。SCSIローダによる初期設定値では，SCSIは内蔵のものを使用し，IDは7となります。

$ED0071番地は，SCSIインタフェースにSASIディスクを接続することを考慮したものです。SASIディスクを接続したときにはSASIディスクのID番号に相当するビットを'1'にします。SCSIローダによる初期値はすべて'0'，すなわち，SASIディスクは接続されていないという設定になります。

❷·4　SCSI装置のメディアバイト

SCSIはハードディスクだけでなく，光磁気ディスクなど，さまざまな種類のデバイスが接続できる可能性があります。これに対応し，現在，SCSI装置のメディアバイトとして，次の4種類が予約されています。

・$F7　ハードディスク

・$F6　光磁気ディスク
・$F5　CD-ROM
・$F4　DAT

CD-ROM と DAT は現在('92 年 2 月現在)，まだ正式なサポートは表明されていませんが，将来を見越して番号は予約されています。

❷·5 SCSIデバイスパラメータ

SCSI 装置の先頭セクタには，そのデバイスがイジェクト可能であるか否かなどの情報を集めたデバイスパラメータと呼ばれる 16 バイトのデータが書き込まれます。この内容は図 9 のようになっています。

●図……9　SCSI デバイスパラメータの内容

先頭からのオフセット	内　容	備　　　考
$00 $01 $02 $03 $04 $05 $06 $07	$58 $36 $38 $53 $43 $53 $49 $31	文字列 〝X68SCSI1〟
$08 $09	BLEN(上位) BLEN(下位)	1セクタのバイト数
$0A $0B $0C $0D	BLOCK Num(上位) 〃 〃 〃(下位)	使用可能な論理ブロック数
$0E	RW	$00以外：SCSI拡張リード/ライトコマンド使用可 $00　　：　〃　不可
$0F	EJ	$00以外：EJECT（メディア交換）可 $00　　：　〃　不可

❷·6 SCSIハードディスクの管理情報

SCSI ハードディスクには，先頭からデバイスパラメータやパーティション情報などが書き込まれます。この内容は図 10 のようになっています。Human 68 K では，ディスク管理の単位が 1 K バイトに固定であるため，表のセクタ値も 1 K バイト単位となっています。SCSI ドライバは，ディスクの 1 ブロックが 256 バイトや 512 バイトの場合には 4 つないし 2 つをまとめて 1 K バイト単位で扱います。

●図……10 SCSI ディスクの管理情報

セクタ番号*	内 容
$00	SCSI デバイスパラメータ
$01	第 1 IPL
$02	パーティション管理情報
$03～$1F	SCSI ディスクドライバ予約領域
$20	第 2 IPL
$21	第 1 FAT（大きさは容量によって変わる）
?	第 2 FAT（ 〃 ）
?	ルートディレクトリ
?	データエリア

*：OS管理上のセクタ（1セクタ＝1Kバイト）を単位とする

❷·7 SCSIコントローラとDMA

SPC とのデータやコマンドの転送には DMA が使用できるようになっていますが，SCSI ドライバでは，この DMA 転送に DMAC のチャンネル# 1 を使用しています。このチャンネルは，従来機種の SASI 用 DMA チャンネルと共用になっていますので，SASI と SCSI の両方を使用するような場合には DMA の設定に注意が必要です。

SASI インタフェースでは，SASI の REQ 信号が DMAC の DREQ (DMA 転送要求) に接続されており，DMAC は DREQ 信号を受け付けると，CPU からバスの使用権を譲り受け，おもむろに転送を開始するという，ごく普通の方法で行っています。ところが，SCSI インタフェースは少し変わった方法を使用しています。

SPC は，DMA 転送要求信号を持っているのですが，X 68000 では，これを DMAC には接続しておらず，DMAC は通常のメモリーメモリ間転送にプログラムします。このままでは，DMA と SPC の転送要求の同期がとれませんので，DMA はまるで意味のないデータを引き取ってしまうことになります。そこで，X 68000 の SCSI インタフェースでは，SPC の DMA

転送要求信号を DTACK (Data Transfer Acknowlege) 信号の作成に使用することで，DMA 転送要求が発生するまで DMA を待たせてしまう方法をとっています。

DTACK 信号というのは，CPU や DMA がアクセスにきたときに，アクセスされた側がデータ転送の完了を示す信号で，通常は周辺デバイスがアクセス速度についていけないときに CPU や DMA を待たせるために使用される信号です。SCSI インタフェースでは，SPC の DMA 要求信号を，この DTACK 信号の作成に使用し，DMA 転送要求がくる前にアクセスされると，DMA 転送要求が発生するまで動作を停止させてしまうようにしているのです。ただ，このようにすると，プログラムのミスなどで DMA 転送要求が発生しないようになると，そのままハングアップしてしまいますので，約 8 μs たっても SPC からの DMA 転送要求が発生しないと，バスエラーを発生させて強制的に回復させるようにしています(DMAC はバスエラーが返されると転送を停止します)。

イメージとしては，DMAC がアクセスにくるとそれをつかまえておき，SCSI バスからデータがくるとそれを引き取らせて手を離す感じです。ただ，つかまえたままにしておくと，だれも動けなくなってしまうので，一定期間 (8 μs) たってもデータがこないようなら，エラーとして転送を中断させるわけです。

● 3 SPC(SCSI プロトコルコントローラ)

SASI は専用 LSI と呼べるものがないため，インタフェースはたんなる I/O ポートにすぎませんでしたが，SCSI は ANSI での規格化がはかられたこともあり，いくつもの専用 LSI がつくられています。X 68000 では，SCSI コントローラ LSI として，富士通の MB 89352 (SPC：SCSI プロトコルコントローラ) が使用されています。この LSI は，SCSI バス制御に必要な機能の多くをハードウェア化しており，ソフトウェアによるバス動作管理の手間がかなり軽減されるようになっています。

❸・1 SPC のレジスター覧

SPC のレジスタのアドレス配置を 471 ページの図 11 に示します。

これらのレジスタのおおまかな機能は次のようになっています。

●図……11　SCSI コントローラ　レジスタ一覧

アドレス	READ/WRITE	bit 7	6	5	4	3	2	1	bit 0	レジスタ名称
+$1	R	ID #7	ID #6	ID #5	ID #4	ID #3	ID #2	ID #1	ID #0	BDID (Bus Device ID)
	W						ID			
+$3	R/W	Reset & Disable	Control Reset	Diag Mode	Arbitration Enable	Parity Enable	Select Enable	Reselect Enable	Interrupt Enable	SCTL (SPC Control)
+$5	R/W	Command Code			RST OUT	Intercept Transfer	Transfer Modifire			SCMD (SPC Command)
+$9	R	Selected	Reselected	Dis-Connected	Command Complete	Service Required	Timeout	SPC Hard Error	Reset Condition	INTS (Interrupt Sense)
	W	(Reset Interrupt:ビット配置はRead時と同じ)								
+$B	R	REQ	ACK	ATN	SEL	BSY	MSG	C/D	I/O	PSNS (Phase Sense)
	W	Diag REQ	Diag ACK	Xfer Enable		Diag BSY	Diag MSG	Diag C/D	Diag I/O	SDGC (SPC Diag Control)
+$D	R	Connected. INIT	TARG	SPC Busy	Transfer in Progress	SCSI Resetin	TC=0	DREG Status Full	Empty	SSTS (SPC Status)
+$F	R	Data Error SCSI	SPC	Xfer Out	'0'	TC Parity Error	'0'	Short Transfer Period	'0'	SERR (SPC Error Status)
+$11	R/W	Busfree INT Enable	'0'				Transfer Phase MSG	C/D	I/O	PCTL (Phase Control)
+$13	R	'0'				MBC				MBC (Modified Byte Counter)
+$15	R/W	Data								DREG (Data Register)
+$17	R	Temporary Data								TEMP (Temporary Register)
	W	Temporary Data								
+$19	R/W	Transfer Counter (上位)								TCH (Transfer Counter High)
+$1B	R/W	Transfer Counter (中位)								TCM (Transfer Counter Mid)
+$1D	R/W	Transfer Counter (下位)								TCL (Transfer Counter Low)

ベースアドレス:SCSIインタフェースボード (CZ-6BS1)……$EA0000
SCSI 内蔵モデル　……$E96020

・BDID レジスタ

自分の ID 番号の設定/読み出しを行います

・SCTL レジスタ

SPC の動作モードや付属機能を使うか否かの選択を行います

・SCMD レジスタ

SPC に対する動作コマンドの指示やデータ転送モードの選択を行います

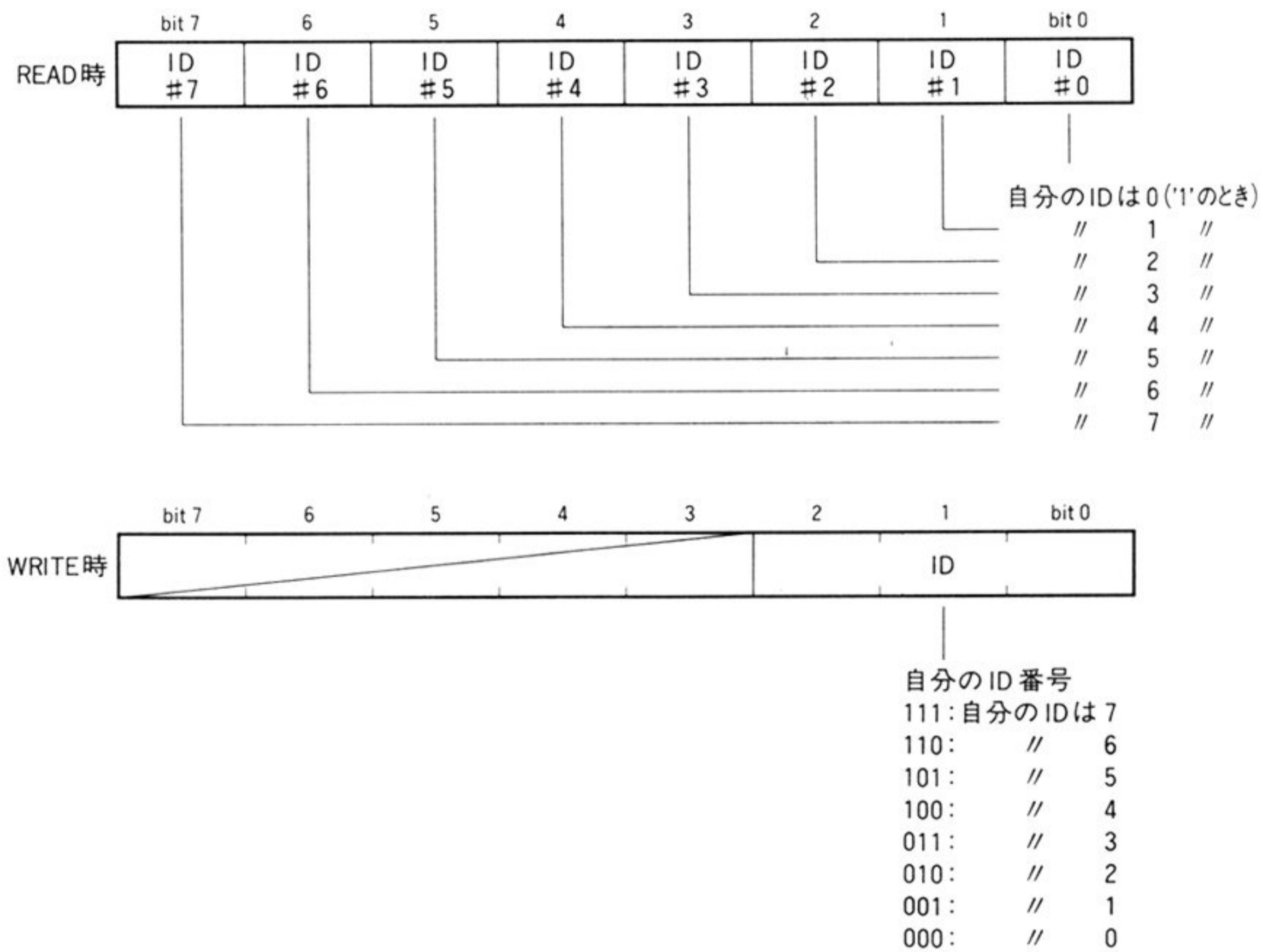
・INTS レジスタ
SPC の割り込み要因の判別や割り込み要因のリセットを行います
・PSNS レジスタ
SCSI バスの制御信号の状態が読み出されます
・SDGC レジスタ
SPC の自己診断用です。通常は使用しません
・SSTS レジスタ
SPC と SCSI バスの間の接続状態や SPC 内部のバッファの状態などが読み出されます
・SERR レジスタ
パリティエラーや SPC のハード的な異常が発生したときのエラーステータスです
・PCTL レジスタ
CPU が SPC に対して，次にどのフェーズで動作するつもりであるのかを明示するのに使用します
・MBC レジスタ
SPC 内部のバッファと CPU とのデータ転送数を制御するカウンタです。初期値は TCL レジスタの下位 4 ビットがセットされます
・DREG レジスタ
DMA による転送を行うときは，このレジスタを通じてデータ転送を行います。このレジスタは 8 バイトの FIFO（First In First Out）バッファとなっています
・TEMP レジスタ
SPC は，転送作業のほとんどを LSI で自動的に行うハード転送モードのほか，SASI インタフェースのように，SCSI の信号をチェックしながら信号の制御を行うマニュアル転送モードを持っています。このマニュアル転送のときに SCSI とのデータ転送に使用するのが TEMP レジタです
TEMP レジスタはこのほか，アービトレーション/セレクションのときに出力する ID 設定用のレジスタとしても使用されます
・TCH/TCM/TCL（転送バイトカウンタ）レジスタ
3 バイト（24 ビット）の転送バイト数カウンタです。ハード転送のときに SCSI 上で 1 バイトの転送が行われるごとにデクリメントされ，転送すべき残りバイト数を保持するほか，セレクションフェーズのときのタイムアウト時間設定用のレジスタとしても使用されます

次に，それぞれのレジスタの中身をもう少し詳しく見ていくことにしましょう。

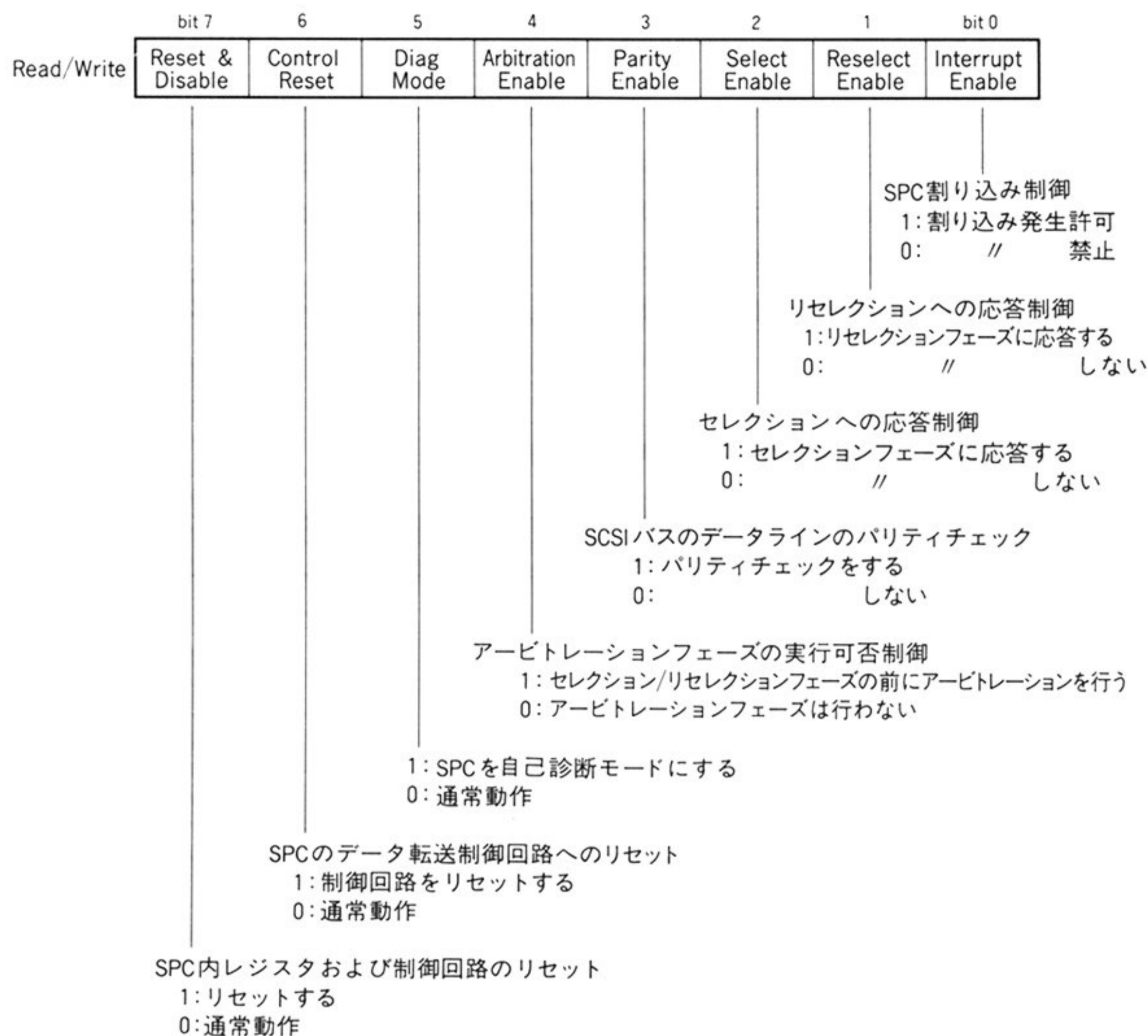
❸·2 BDIDレジスタ

ビット配置は図12のようになっています。書き込み時は，レジスタの下位3ビットで0～7までのID番号を設定します。読み出し時は，設定したID値に対応するビットだけが'1'になったデータが読み出されます。たとえば，IDとして$07を書き込むと，ビット7だけが'1'になったデータ，すなわち，$80が読み出されます。

このレジスタから読み出されるデータは，アービトレーションフェーズでバス上に出力されるものと同じです。

●図……12　BDIDレジスタ（ベースアドレス＋$01）

❸·3 SCTLレジスタ

ビット配置は474ページの図13のようになっています。

●図……13　SCTL レジスタ（ベースアドレス＋$03）

各ビットの意味は次のようになっています。

bit 7：Reset & Disable

SPC全体のリセット信号に相当します。'1'を書き込むとリセットされます。ハードウェアリセット時（電源ON直後や本体のRESETスイッチが押されたとき）にも，このビットは'1'に設定されます。SPCは，SCSIバスと完全に切り離された状態になり，外部からのセレクションなどには応答しなくなります。リセット後，SPCを使用しはじめる前に，このビットを'0'にしなくてはなりません。

bit 6：Control Reset

SPC内部のデータ転送制御回路だけをリセットします。'1'を書き込むと，'0'に戻すまでリセ

ットしたままとなります。このビットを'1'にしても，SCSIとの結合関係には変化はありません。

bit 5：Diag Mode

SPCを自己診断モードにするためのもので，このビットを'1'にすると，自己診断モードになります。このモードでは，SPCはSCSIと完全に切り離され，かわりにSDGCレジスタへの設定値がSCSIバスの状態であるかのように動作します。

bit 4：Arbitration Enable

セレクション/リセレクションフェーズの前にアービトレーションフェーズを実行するか否かを選択します。'1'のときにはアービトレーションフェーズが実行され，'0'のときにはSASIと同様，アービトレーションフェーズを省略してセレクションフェーズを実行します。

bit 3：Parity Enable

SCSIバスのデータラインのパリティチェックを行うか否かを選択します。この設定は，SPCがデータを受け取るときにチェックを行うか否かを設定するものです。SPCがデータを出力するときのパリティの生成は，このビットの設定に関係なく，無条件に実施されます。X 68000では，このビットを'0'にしておきます。

bit 2：Select Enable

セレクションフェーズに対してターゲットとして応答するか否かを選択します。'1'にするとセレクションフェーズに応答し，'0'のときは無視します。このビットは，自分がターゲットとして動作するか否かを選択するものであると考えてよいでしょう。X 68000は通常イニシエータとしてしか動作しませんので，このビットは'0'に設定します。

bit 1：Reselect Enable

リセレクションフェーズに応答するか否かを選択します。'1'に設定すると，リセレクションフェーズに対してイニシエータとして応答し，'0'のときは無視します。SCSIのフェーズ遷移のところで説明したディスコネクト/リコネクト機能を使用する場合には，このビットを'1'に設定します。X 68000のSCSIインタフェースのような，イニシエータが1つしかないようなシステムでは，ディスコネクト/リコネクト機能を使っても，バス使用効率の向上には貢献しないためか，X 68000を立ち上げた後で，このビットを見ると，'0'になっています。

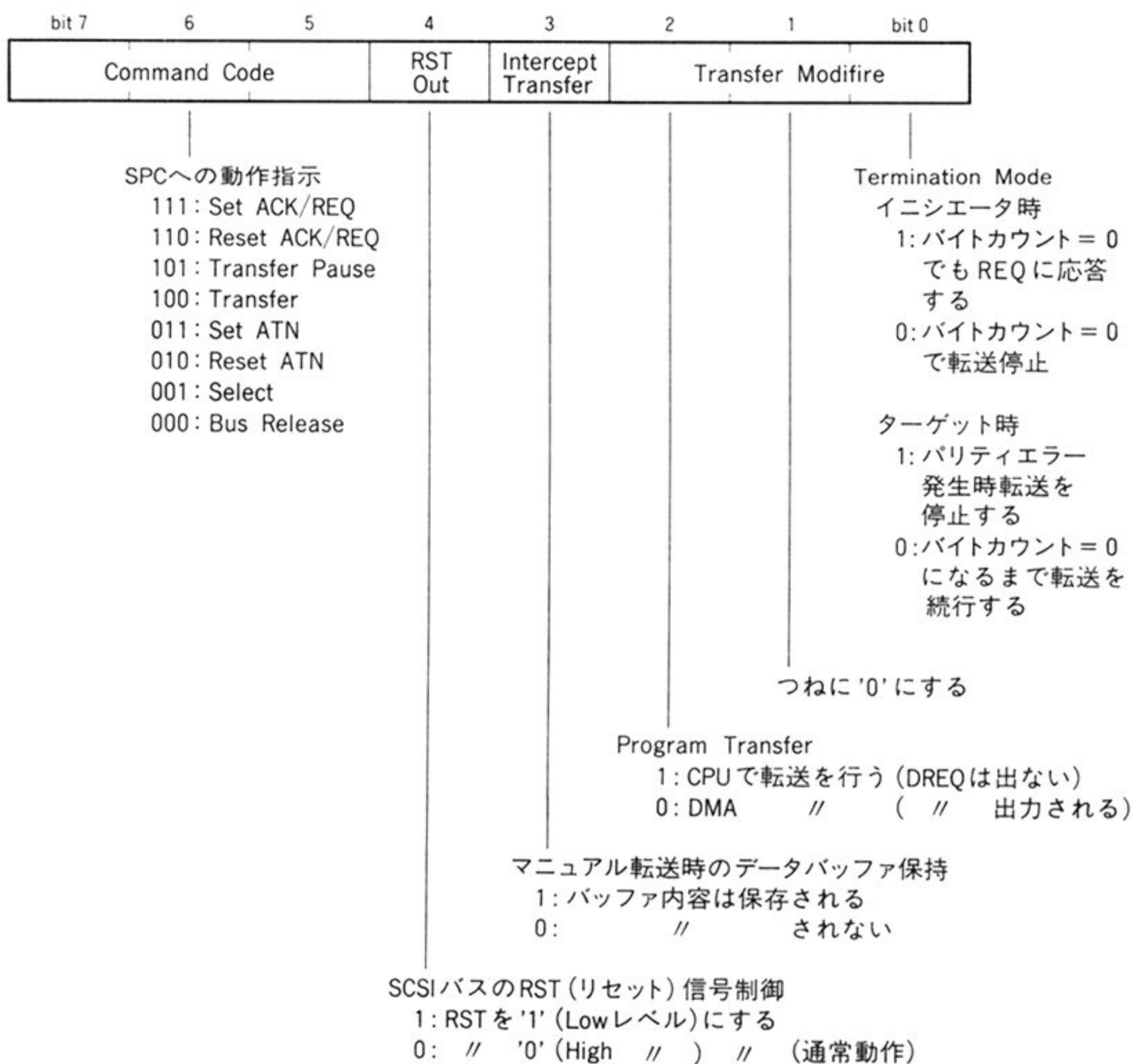
bit 0：Interrupt Enable

SPCからの割り込みの許可/禁止の制御を行うビットです。'1'のときに割り込み発生が許可に，'0'のときには禁止になります。このビットを'0'にしても，SCSI上のResetコンディション（RST信号が'1'になる）が検出された場合には割り込みが発生します。

また，このビットが'0'であっても，割り込み要因はINTSレジスタに反映されます。

❸·4 SCMDレジスタ

ビット配置は図14のようになっています。

●図……14 SCMDレジスタ（ベースアドレス＋$05）

それぞれのビットの意味は次のようになっています。

bit 7, 6, 5 : Command Code

SPCへの動作実行指示を行います。それぞれのコマンドの動作については後で説明します。

bit 4 : RST Out

'1'を書き込むとSCSIバスのRST信号を'1'にし，SCSIバスをリセットします。SCTLレジスタが'1'のときは，このビットの操作は無効です。

bit 3：Intercept Transfer

このビットを'1'にしてからマニュアル転送（CPU で REQ/ACK 信号などを制御するモード）を行うときは，SPC 内部にある 8 バイトの FIFO バッファの内容は保存されます。

bit 2：Program Transfer

'1'にすると DREQ (DMA 転送要求) 信号を出力しないモードになります。マニュアル転送のときには，このビットを'1'にしたほうがよいでしょう。前にも述べたとおり，X 68000 では DREQ 信号を DTACK 信号の作成に使用しています。このビットを'1'にすると DREG へのアクセスができなくなってしまう（すべてバスエラーになる）ので，ハード転送を行うときには，このビットは必ず'0'にして DREQ 信号を発生させるようにしてください。

bit 1：(未使用)

使用されていません。つねに'0'を設定するようにしてください。

bit 0：Termination Mode

イニシエータとして動作しているときと，ターゲットとして動作しているときとで意味が変わります。

イニシエータとして動作しているとき，このビットが'0'になっていると，転送バイトカウンタの値が 0 になった時点で転送動作は停止します。'1'の設定は，データイン/データアウトフェーズのときだけ有効です。このとき，転送バイトカウンタが 0 になっても，同一フェーズのままターゲットから REQ 信号がくれば応答します。データの方向がターゲットからイニシエータ側の場合は取り込んだデータは破棄され，イニシエータからターゲットの場合は$00 が送られます。この転送動作を Padding 転送と呼びます。

転送カウンタを 0 にしたまま転送動作に入ると，最初の転送から Padding 転送になります。このとき，Transfer コマンドの発行の前に TEMP レジスタに$00 を書き込むようにしてください。

ターゲットとして動作しているときに，このビットが'1'になっていると，転送中にパリティエラーを検出した場合，ただちに転送を終了しますが，'0'になっていると，パリティエラーを検出しても転送カウンタが 0 になるまで転送を続行します。

❸·5 INTSレジスタ

ビット配置は 478 ページの図 15 のようになっています。

割り込み要因となる条件が成立すると，割り込み発生の許可/禁止 (SCTL レジスタのビット 0) に関係なく，INTS レジスタの該当ビットは'1'にセットされます。CPU が，このレジスタに書き込み動作を行うと，'1'を書き込んだビットだけが'0'にクリアされます。それぞれの割り

●図……15　INTS レジスタ（ベースアドレス＋$09）

	bit 7	6	5	4	3	2	1	bit 0
READ/WRITE	Selected	Reselected	Disconnected	Command Complete	Service Required	Time Out	SPC Hard Error	Reset Condition

bit 0（Reset Condition）
1: SCSIバスがReset
　された
0: 通常動作

bit 1（SPC Hard Error）
1: SERRレジスタに示される
　エラーのうち，
　・TC Parity Error
　・Short Transfer Period
　が検出された
0: 通常動作

bit 2（Time Out）
1: セレクション，またはリセレクション実行後
　一定期間たっても相手からの応答が
　ない
0: 通常動作

bit 3（Service Required）
1: イニシエータとして動作中に
　バス上のフェーズとPCTLレジスタで指定される
　フェーズが一致しないため，Transferコマンドを
　実行できない（できなくなった）
0: 通常動作

bit 4（Command Complete）
1: SelectコマンドやTransferコマンドの実行が完了した
0: 通常動作

bit 5（Disconnected）
1: SCSI上でバスフリーフェーズが検出された（SCTLレジスタのbit 7='1'のとき）
0: 通常動作

bit 6（Reselected）
1: SCSI上の他のデバイスからリセレクションフェーズによって選択された
0: 通常動作

bit 7（Selected）
1: SCSI上の他のデバイスからセレクションフェーズによって選択された
0: 通常動作

込み要因は次のようになっています。

bit 7：Selected

セレクションフェーズによって，SPCが他のイニシエータと接続されたことを示します。この割り込みが発生して以降，Bus Releaseコマンドが発行されたり，SCSIバスがリセットされるまで，SPCはターゲットとして動作したままとなります。

bit 6：Reselected

リセレクションフェーズによって，SPCがコントローラと再接続されたことを示します。これ以降，Disconnect割り込みが発生するか，SCSIバスがリセットされるまで，SPCはイニ

シエータとして動作しつづけます。

bit 5：Disconnect

バスフリー割り込み許可（PCTLレジスタのビット7が'1'）のとき，バスフリーフェーズが検出されると'1'になります。このビットが'1'になっていると，SPCはSCSI上での動作を行いませんので，SCSIバスを使用する前に必ずリセットしておかなくてはなりません。

bit 4：Command Complete

Selectコマンドや Transferコマンドの処理が終了したことを示します。SPCがターゲットとして動作しているときに，パリティエラーによって転送が停止した場合にも，このビットが'1'になります。

bit 3：Service Required

イニシエータとして動作中に，PCTLレジスタの下位3ビットで行っているフェーズとバス上のフェーズが一致しないために転送が実行できなかったり，転送中にフェーズが一致しなくなり（ターゲットがフェーズを切り替えてしまったとき），転送が中断されてしまったことを示します。このようなとき，CPUは状況を判断して適宜回復措置をとらなくてはなりません。若干注意が必要なのは，転送中にフェーズが一致しなくなってしまった場合で，このとき，SCSIバス上の転送動作はただちに中断しますが，SPC内部バッファのデータは残ったままになっている可能性があります。データ入力時にはSPC内部バッファのデータがすべて引き取られるまで，また，出力時には内部データバッファへのデータ先取りシーケンスが終了するまで，SPCの転送動作は終結しませんので注意が必要です。このビットが'1'になってしまった場合は，SSTSレジスタを見てSPCの転送動作状態を確認するようにしてください。

bit 2：Time Out

Selectコマンドによるセレクション/リセレクションフェーズが行われたにもかかわらず，一定期間たっても相手が応答しなかったことを示します。これをセレクションタイムアウトと呼ぶこともあります。

セレクションタイムアウトが発生した場合，SPCはSEL信号を'1'にしたままにしてしまいます。この状態は，TEMPレジスタに$00を書き込むことで復旧できます。セレクションタイムアウトが発生した後，バスを解放するのはこの方法で行ってください。

bit 1：SPC Hard Error割り込み

SPCがTC Parity ErrorやShort Transfer Periodエラー（いずれもSERRレジスタに反映されます）を検出したことを示します。この割り込みが発生しても，SPCは実行中の動作を停止することはありません。

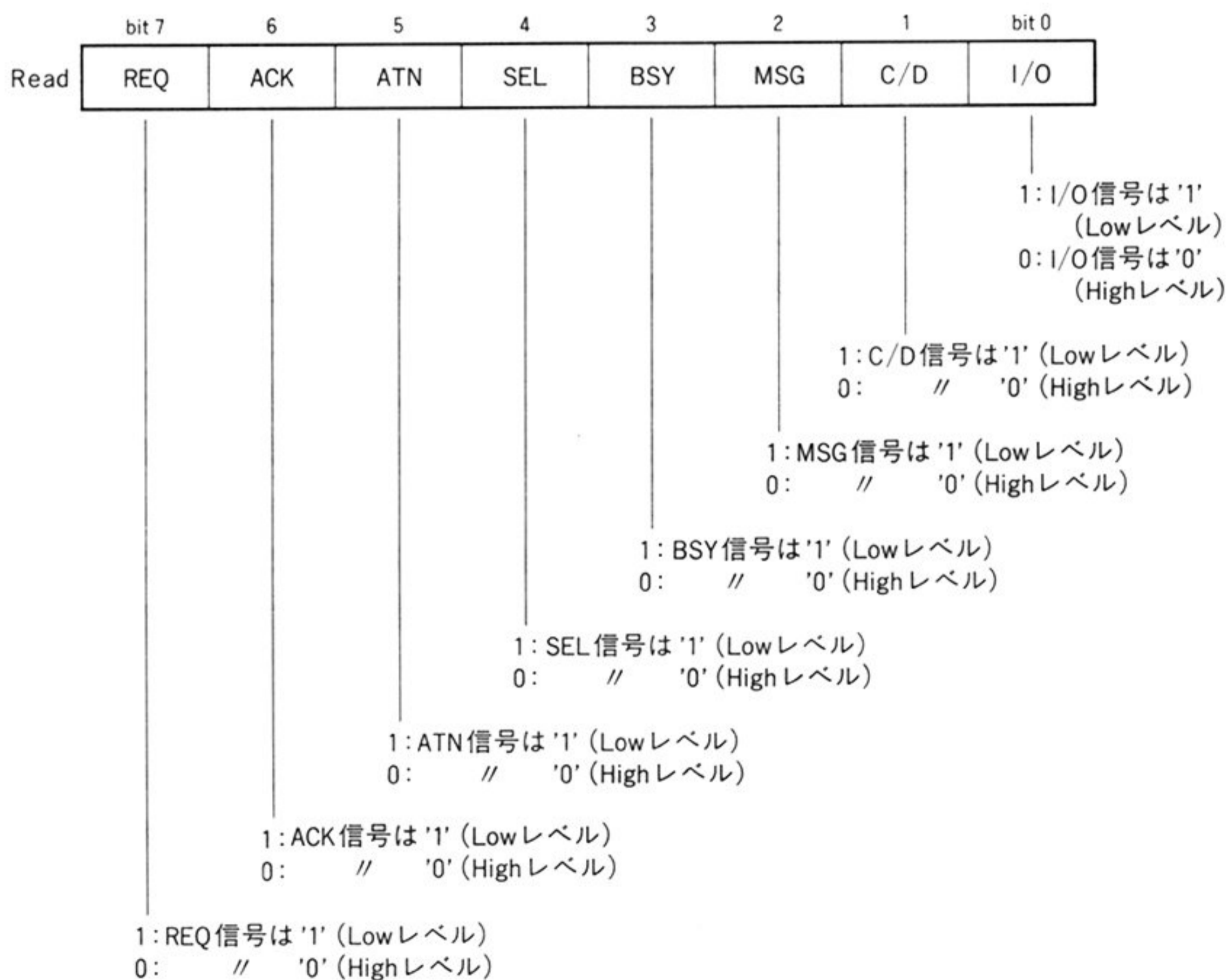
bit 0：Reset Condition割り込み

SCSIバス上にリセットがかかった（RST信号が'1'になった）ことを示します。RST信号の継続時間は規定がありませんので，この割り込みのリセットはRST信号が'0'に戻った

(SSTSレジスタのビット3が'0'になる)のを確認してから行う必要があります。SCSIバスのリセットがかかると，実行中のバス動作はすべて打ち切られ，バスフリーフェーズになります。SPCの内部状態もリセットされますが，BDID，SCTL，SCMD，PCTL，転送バイトカウントの各レジスタの内容は変化しません。

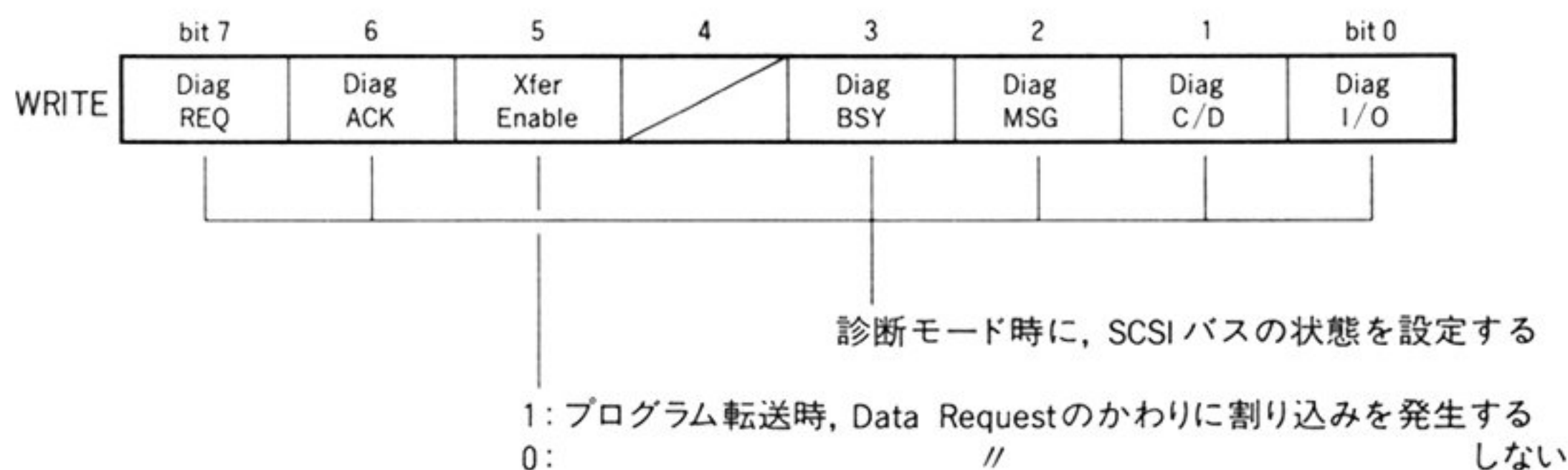
❸·6 PSNSレジスタ

SCSIバスの状態が読み出されます。ビット配置は図16のようになっています。このレジスタはSPCの動作状態に関係なく読み出すことができます。読み出されるデータとSCSIバス上の状態の関係は，SASIインタフェースポート同様，'1'のときにSCSIバス上はLowレベルとなっています。

●図……16　PSNSレジスタ（ベースアドレス＋$0B）

❸·7 SDGCレジスタ

ビット配置は図17のようになっています。ビット5は，転送を実施するときにデータ転送要求(Data Request)割り込みを発生するか否かを選択するビットで，'1'のときに割り込み発生を許可します。

ビット5以外はSPCの自己診断のときに使用されます。SPCを自己診断モードにしたとき(SCTLレジスタのビット5を'1'にしたとき)，SPCのSCSIバスインタフェース信号はSCSIバスと切り離され，SDGCレジスタにセットした値がSCSIバス上の状態であるかのように動作します。これによってSPCの動作チェックをすることができるわけです。自己診断モードでのアービトレーションはつねに成功します。

●図……17　SDGC レジスタ（ベースアドレス＋$0B)

❸·8 SSTSレジスタ

ビット配置は482ページの図18のようになっています。

このレジスタはSPCの動作に関係なく，いつでも読み出すことができます。各ビットの意味は次のようになっています。

bit 7, 6：Connected

SCSIバスとの結合状態を示します。イニシエータとして結合しているとビット7が，ターゲットとして結合しているとビット6が'1'になります。

bit 5：SPC Busy

SPCがコマンドの実行中ないし実行待ち状態であることを示します。

bit 4：Transfer In Progress

●図……18　SSTSレジスタ（ベースレジスタ＋$0D）

bit 7	bit 6	bit 5	bit 4	動　作　状　態
0	0	0	0	SCSIと非結合中。SPCは実行コマンドを保持していない
0	0	1	0	SCSIと非結合中。Selectコマンド保持中（バスフリー待ち/アービトレーション中）
0	1	0	0	ターゲットとして動作中（SCSI上で実行中の動作なし/マニュアル転送中）
0	1	1	0	SCSI上でリセレクションフェーズ実行中
0	1	1	1	ターゲットとして動作中（ハード転送実行中）
1	0	0	0	イニシエータとして動作中（SCSI上で実行中の動作なし/マニュアル転送中）
1	0	0	1	イニシエータとして動作中（REQ信号はきているが，転送は実行されていない）
1	0	1	0	SCSI上でリセレクションフェーズを実行中
1	0	1	1	イニシエータとして動作中（ハード転送実行中）

ハード転送が実行中であるか，または SCSI 上で転送フェーズが要求されていることを示します。

bit 3：SCSI Reset In

SCSI のリセット信号（RST）の状態を示します。

bit 2：TC=0

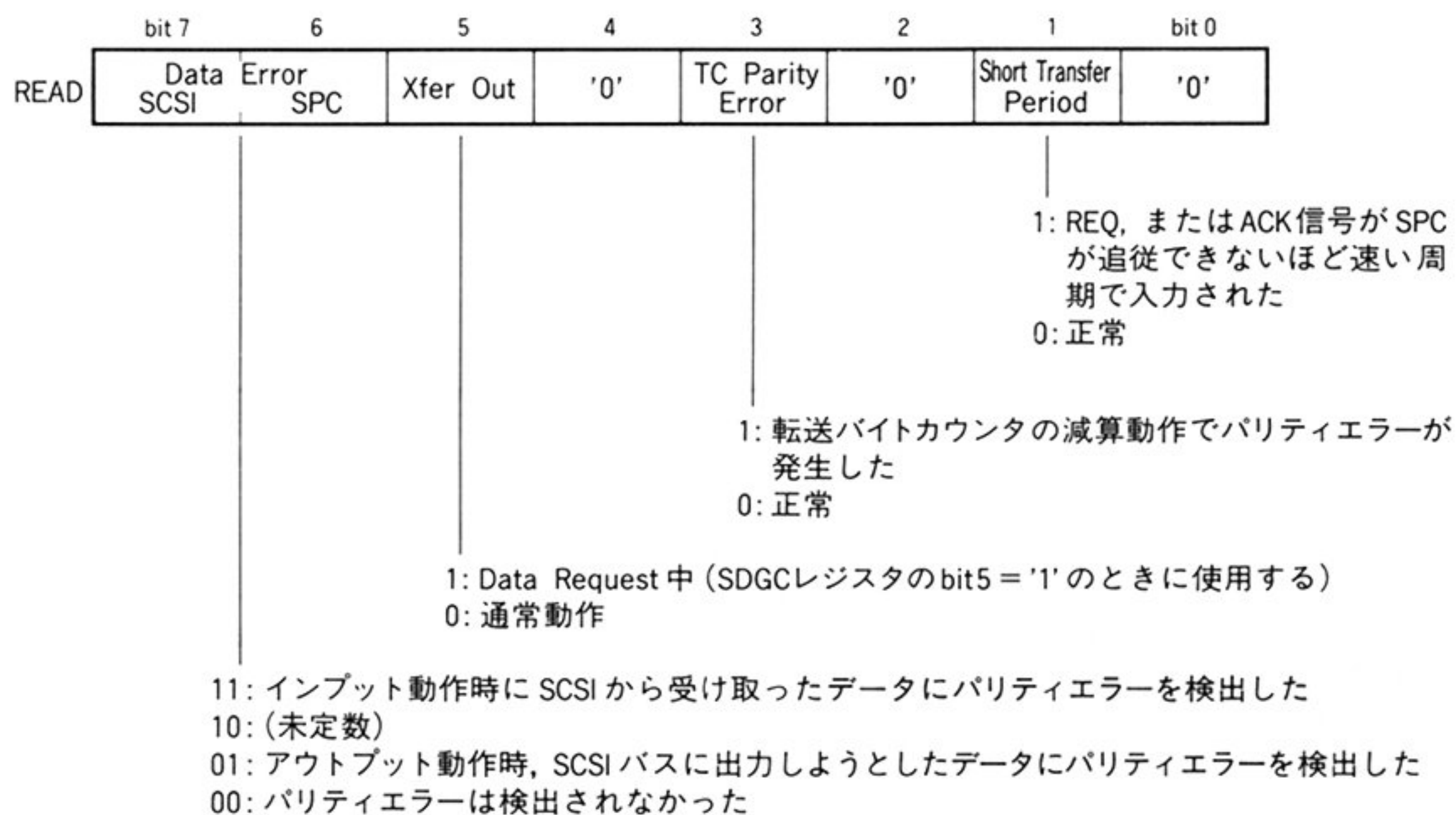
転送バイトカウンタ（TCH，TCM，TCL レジスタ）の値が 0 になったことを示します。

bit 1, 0：DREG Status

SPC 内部の FIFO バッファの状態を示します。SPC の FIFO バッファは 8 バイトあり，中にデータが入っていないとビット 0 が，データがフル（8 バイト入っている）ならビット 1 が '1' になります。両方とも '0'になっているときは，空でもフルでもないということですから，1 〜 7 バイト分のデータが入っている状態ということになります。

❸·9 SERRレジスタ

ビット配置を図 19 に示します。

●図……19 SERRレジスタ（ベースアドレス＋$0F）

このレジスタに示されるエラーのうち，ビット3かビット1のいずれかが'1'になると，SPC Hardware Error (INTSレジスタのビット1）となり，割り込みが発生します。それぞれのエラーの意味は次のようになっています。

bit 7, 6 : Data Error

SCSI上でパリティエラーを検出したことを示します。ビット6と7の組み合わせとその内容は図19に示したとおりですが，読みかえると，ビット6はSPCがパリティエラーを発生したときに'1'となり，ビット7は入力時に検出した場合に'1'，出力時に検出したときには'0'になると考えればよいようです。

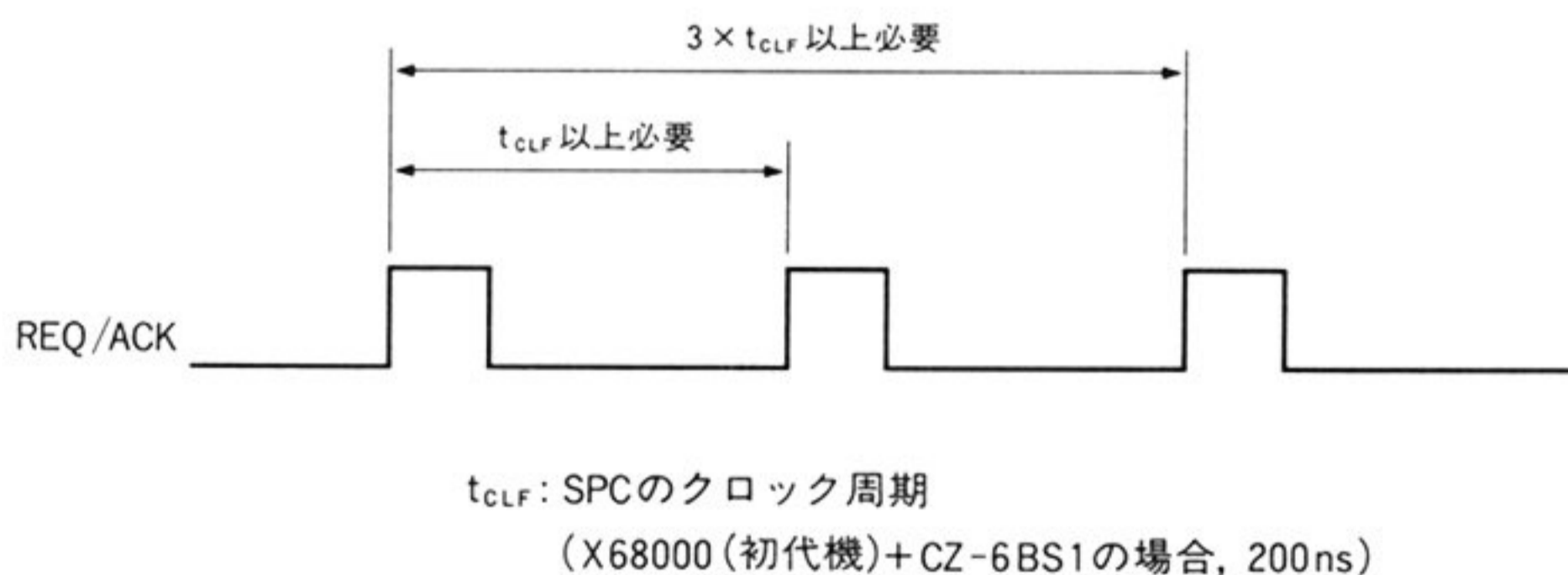
bit 3 : TC Parity Error

SPCが転送バイトカウンタのデクリメント動作をしているときにパリティエラーを検出したことを示します。

bit 1 : Short Transfer Period

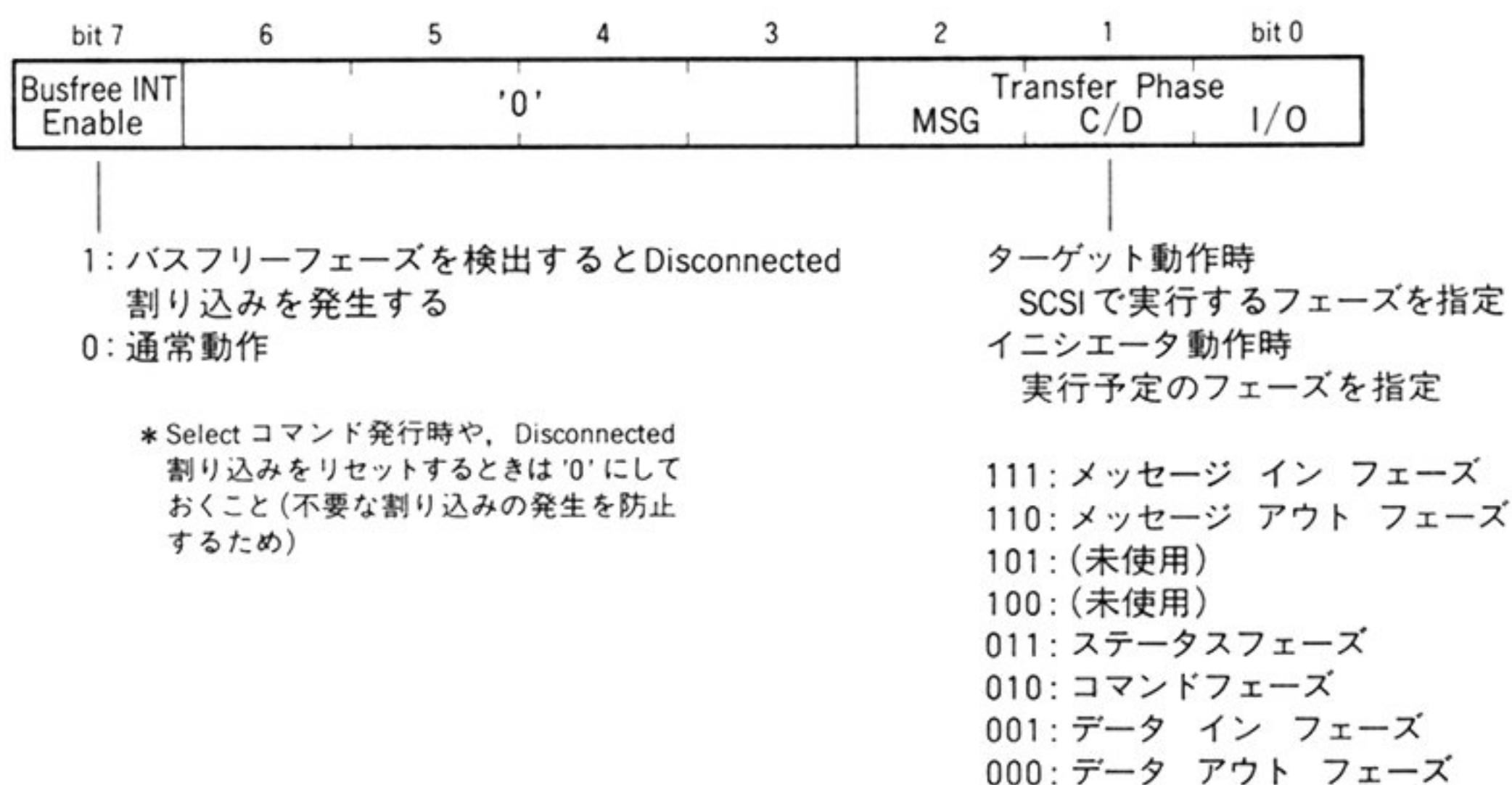
REQやACK信号入力がSPCが追従できないほど速い周期で入力されたことを示します。SPCが追従できる周期を図20に示します。このタイミングはSPCに与えられているクロック周波数をもとにして算出されます。クロック周波数は取り扱い説明書などを見ても書いてありませんでしたので，実測したところ(CZ-6BS1を初代機に入れた場合)，5 MHzでした。これより，t_{CLF}=200 nsとなります。

●図……20　REQ/ACK信号周期の制限

t_{CLF} : SPCのクロック周期
(X68000(初代機)+CZ-6BS1の場合，200ns)

❸·10 PCTLレジスタ

ビット配置は485ページの図21のようになっています。

ビット6～3は使用されていませんが，'0'を書き込むようにしてください。おのおののビットの意味は次のようになっています。

●図……21 PCTLレジスタ（ベースアドレス＋$11）

bit 7：Busfree INT Enable

バスフリーフェーズ検出による Disconnected 割り込みを発生するか否かを選択します。'1'にすると割り込み発生許可になります。Select コマンドを発行するときや，Disconnected 割り込みをリセットするときには，このビットを必ず'0'にして不要な割り込みの発生を禁止してください。

bit 2, 1, 0：Transfer Phase

イニシエータとして SCSI バスと結合しているときには実行しているつもりのフェーズを，ターゲットとして結合しているときには SCSI で実行するフェーズを設定します。イニシエータとしてハード転送を行う場合，バス上のフェーズとこのレジスタで指定したフェーズが一致しないと，転送動作が行われませんので注意してください。

Select コマンドを発行するときには，このレジスタのビット 0 が'0'だとセレクションフェーズ，'1'だとリセレクションフェーズの指定になります。

●4 SPCの転送モード

SPC の持つ転送モードを 486 ページの図 22 にまとめておきました。

マニュアル転送は，REQ-ACK ハンドシェークの制御などをすべて CPU でコントロール

●図……22　SPCの持つ転送モード

<table>
<tr><th colspan="2">転送モード</th><th>データアクセス</th><th>DREQ信号</th><th>CPUが転送制御に使用するレジスタ</th><th>備　考</th></tr>
<tr><td colspan="2">マニュアル転送</td><td>TEMPレジスタ</td><td>出力しない</td><td>PSNSレジスタ</td><td></td></tr>
<tr><td rowspan="2">ハード転送</td><td>プログラム転送</td><td>DREGレジスタ</td><td>出力しない</td><td>SSTSレジスタ
(または割り込み)</td><td>X 68000では使用不可</td></tr>
<tr><td>DMA転送</td><td>DREGレジスタ</td><td>出力する</td><td></td><td></td></tr>
</table>

するものです。SASIポートと似たようなものだと思えばよいでしょう。このモードでは，SCSIバスのデータラインのアクセスはTEMPレジスタを通して行います。ハード転送は，このような面倒な制御のほとんどをSPC自体で行ってしまうモードです。このモードでは，SCSIバスのデータラインとのアクセスはDREGレジスタで行い，8バイトのFIFOバッファが有効となります。

さらに，ハード転送モードはDREQ（DMA転送要求）信号を出力するか否かによって，DMA転送モードとプログラム転送モードの2つに分類できます。ただし，X 68000のSCSIインタフェースでは，SPCのDREQ信号をDTACK信号を作成するのに使用しているため，プログラム転送モードを選択すると，DREGレジスタへのアクセスができなくなります（すべてバスエラーになってしまいます）。

●5 SPCのコマンド

SCMDレジスタの上位3ビットに書き込むコマンドと，その動作は次のようになっています。

5·1 Bus Releaseコマンド

ターゲットとして動作しているときにバスフリーフェーズへの移行を行うときに使用します。データ転送中から移行するときには，Transfer Pauseコマンドでデータ転送を停止させてから行うようにしてください。

このコマンドはSelectコマンド発行後，バスフリーフェーズ待ちの状態にあるときに

Select コマンドをキャンセルするために使用することもできます。

❺·2 Selectコマンド

セレクション/リセレクションフェーズの起動要求コマンドです。Arbitration Enable になっているとき(SCTL レジスタのビット 4 が'1')には，セレクション/リセレクションフェーズの前にアービトレーションフェーズが自動的に実行されます。アービトレーションフェーズで負けても，このコマンドの実行は終了します。

アービトレーションで勝った場合や，Arbitration Enabel でない場合には，セレクション/リセレクションフェーズが実行されます。

Select コマンドが失敗した（セレクションタイムアウトになった）ときには Time Out (INTS レジスタのビット 2)を'1'として，また，セレクションが成功した場合には Command Complete（INTS レジスタのビット 4）を'1'にして割り込みを発生します。

Select コマンドは，コマンド発行前に次の設定を必要とします。

PCTL レジスタのビット 0

セレクションフェーズを実行するのか，リセレクションフェーズを実行するのかを選択します。'0'のときにはセレクションフェーズ，'1'のときにはリセレクションフェーズが実行されます。

Set ATN コマンドの発行

セレクションの後，メッセージアウトフェーズを実行したい場合には，Select コマンドに先立って Set ATN コマンドを発行し，ATN 信号を'1'にするよう SPC に指示します。

TEMP レジスタ

セレクション/リセレクションフェーズのときにデータラインに出力する値（自分と相手の ID）に対応するビットが'1'になったデータをセットします。

TCH/TCM レジスタ

セレクションフェーズ/リセレクションフェーズのときの相手からの応答を待つ時間（BSY 信号が'1'になるまでの時間)を設定します。この時間 T は，TCH/TCM で示される値を X とすると，

$$T=(X\times256+15)\times t_{CLF}\times2$$

で表されます。ここで，t_{CLF}は SPC に与えられているクロックの周期です(X 68000 では 200

nsで計算します)。この時間が経過しても，BSY信号が'1'にならないと，セレクションタイムアウトになります。Xが0のときだけは例外で，監視時間は無限大になります。

TCLレジスタ

SPCが，BSYとSEL信号がともに'0'となってからアービトレーションやセレクション/リセレクションフェーズを開始するまでの時間を設定します。この待ち時間は，TCLへの設定値をXとすれば，$(X+6)\times t_{CLF}$から$(X+7)\times t_{CLF}$の間の値となります。Xの値の範囲は$00～$0Fで，$10以上の設定は禁止されています。X68000の場合，推奨値は$03です。

❺·3 Set ATNコマンド

SCSIバスのATNラインを'1'にします。SPCがイニシエータのときだけ有効です。Selectコマンドの前に発行された場合には，Selectコマンドの実行時にATNラインが'1'になります。ただし，セレクションフェーズ実行前にSelectedかReselected割り込みが発生した場合は，Set ATNコマンドは破棄されます。

❺·4 Reset ATNコマンド

SCSIバスに出力中のATN信号を'0'に復帰させます。ただし，SPCが転送実行中にパリティエラーを検出したことによって自動的にSCSIバスのATN信号を'1'にした場合には，実行中のTransferコマンドが終了するまで，このコマンドでATNをリセットしてはいけません。

マニュアル転送のときにATN信号をリセットするときは，ACK信号を'1'にする前に行ってください。

次の場合には，SPCは自動的にATN信号を'0'に復帰させます。

- Disconnected割り込みが発生したとき
- ハード転送モードで，メッセージアウトフェーズを実行する場合で，最終バイトを送出するとき
- セレクションタイムアウト検出後，BSY信号の応答がないまま，割り込みをリセットしてSPCがSCSIバスと非結合状態になるとき
- セレクションタイムアウト時間を無限大に設定したとき，BSY信号の応答がないままTime Outビット(INTSレジスタのビット2)に'1'を書き込んで非結合状態に復帰させる

とき

❺·5 Transferコマンド

データイン/アウト，ステータス，コマンド，メッセージイン/アウトの各フェーズでのデータ転送（ハード転送）の実行開始を指示するコマンドです。このコマンドを実行する前に，次の設定を行っておく必要があります。

・転送バイトカウンタ（TCH/TCM/TCL）に転送を行うバイト数を設定する
・PCTLレジスタの下位3ビットに実行するフェーズを設定する

ターゲットとして動作しているときには，コマンドの実行は次の条件で終了します。

・転送バイトカウンタに設定したバイト数分の転送が終了した
・Transfer Pauseコマンドが発行された
・SCMDレジスタのビット0を'1'にしたインプット動作のときにデータラインにパリティエラーを検出した

イニシエータとして動作しているときには次の条件でコマンド実行を終了します。

・Padding転送モードでないときに転送バイトカウンタで指定されたバイト数の転送が終了した
・ターゲットがPCTLで指定した以外のフェーズに移行した
・Disconnected割り込みが発生した

転送開始時，PCTLレジスタで指定したフェーズとSCSIバス上のフェーズが一致しないと，転送動作は開始されず，Service Required割り込みが発生します。
なお，イニシエータとしてハード転送を実行するときには，転送バイトカウンタの値は2以上にしてください。

❺·6 Transfer Pauseコマンド

ターゲットとして動作しているとき，実行中のハード転送動作を中断させるコマンドです。イニシエータとして動作しているときには，このコマンドは使用できません。アウトプット動作時，このコマンドを発行した後はDREGレジスタへの書き込みを行ってはなりません。

❺·7 Set ACK/REQコマンド

マニュアル転送時にSCSIバスのACK/REQ信号を'1'にするために使用します。イニシエータとして動作しているときにはACK信号が，ターゲットとして動作しているときにはREQ信号が'1'になります。このとき，PCTLレジスタの下位3ビットには実行するフェーズを設定します。

❺·8 Reset ACK/REQコマンド

マニュアル転送時，SCSIバスのACK/REQ信号を'0'にするために使用します。イニシエータとして動作しているときにはACK信号が，ターゲットとして動作しているときにはREQ信号が'0'になります。必要なら，本コマンドに先行してSet ATNコマンドを発行しておくことで，ATN信号を出力させることができます。

メッセージインフェーズでの転送をハード転送で行った場合，SPCは最終バイトを受け取った後，ACK信号を'1'にしたまま転送を終了してしまいますので，このコマンドでACK信号を'0'に復帰させる必要があります。

●6 SCSIの主要コマンド

SCSIの規格化時にSCSIインタフェースで用いられるコマンドも整理されたのですが，それだけではまだ不十分であるというメーカの声が強いことから，ANSIでもSCSIコマンドの

規格化作業を行っています。これらのコマンドはCCS (Common Command Set) と呼ばれています。

CCSのすべてについて説明するのはとても無理なので，ここではHuman68KのSCSIドライバなどが使用しているコマンドに限定して説明しておくことにしましょう。

❻·1 SCSIコマンドの一般形

SCSIコマンドフォーマットの一般形を492ページの図23に示します。

SCSIコマンドは，SASIと同じ6バイト長コマンドに加え，10バイト長，12バイト長のコマンドがあります。このうち，10バイト長コマンドは，コマンドの最初のバイトの上位3ビット (グループコード) が'001'，12バイト長コマンドは'101'になっています。6バイトコマンドのフォーマットは，名称が変更されている程度で，ほとんどSASIと同じです。

Human68KのSCSIドライバなどが使用するコマンドは，ほとんどがグループ0 (グループコードが'000') で，Read Capacityなど，ごく一部のコマンドがグループ1となっており，グループ5のコマンドはありません。

コントロールバイト(各命令の最終バイト)のLinkビットは，ターゲットに複数コマンドの連続実行をさせるために使用するフラグです。連続実行を行うときには，このビットを'1'にします。ターゲットにこの機能がサポートされていると，コマンド実行後のステータスフェーズでINTERMEDIATEステータスを返し，メッセージインフェーズに続いてコマンドフェーズに移行します。

Flagビットは，Linkビットを'1'にしたときにのみ有効です。Linkビットが'0'のときにこのビットを'1'にしてはなりません。このビットが'1'だと，ターゲットは，コマンドが正常終了した後に，LINKED COMMAND COMPLETE WITH FLAGメッセージを，'0'のときにはLINKED COMMAND COMPLETEメッセージを通知します。通常，このフラグは一連のコマンドの中で特定のコマンドの実行が終了したことを検出するためのマークとして使用します (どちらのメッセージが返ってきたかによって，マークしたコマンドか否かが区別できる)。

●図……23　SCSIコマンドの一般形

6バイト長コマンド(グループ0)

転送順序	bit7	6	5	4	3	2	1	bit0	備　考
0	グループコード			コマンドコード					オペレーションコード
1	LUN (論理ユニット番号)			論理ブロックアドレス(上位)					
2	論理ブロックアドレス								
3	論理ブロックアドレス(下位)								
4	転送長								
5	V	Reserved					Flag	Link	コントロールバイト

10バイト長コマンド(グループ1)/12バイト長コマンド(グループ5)

転送順序 10バイト長	転送順序 12バイト長	bit7	6	5	4	3	2	1	bit0	備　考
0	0	グループコード			コマンドコード					オペレーションコード
1	1	LUN (論理ユニット番号)			(Reserved)				Rel Adr	
2	2	論理ブロックアドレス(上位)								
3	3	論理ブロックアドレス								
4	4	論理ブロックアドレス								
5	5	論理ブロックアドレス(下位)								
6	6	Reserved								(将来拡張用)
	7	〃								
	8	〃								
7	9	転送長(上位)								
8	10	転送長(下位)								
9	11	V	Reserved					Flag	Link	コントロールバイト

Rel Adr:論理ブロックアドレスは最後にアクセスしたところからの相対値である
　（2の補数表記）
V:ベンダ(メーカ)ごとに自由に使用可

❻·2 SCSIコマンドのコード

X 68000で使用される主要なコマンドのコード一覧を 図24に示します。

●図……24 SCSI主要コマンド

コマンドの1バイト目			コマンド名	備考
オペレーションコード	グループコード	コマンドコード		
$00	0	$0	Test Unit Ready	ユニットが使用可能であるか調べる
$01	0	$1	Rezero Unit	シリンダ0へのヘッド移動などを行う
$03	0	$3	Request Sense	センスデータの取得
$04	0	$4	Format Unit	メディアのフォーマットを行う
$08	0	$8	Read	データの読み出し
$0A	0	$A	Write	データの書き込み
$12	0	$12	Inquiry	ターゲットおよびユニットの属性情報取得
$1A	0	$1A	Mode Sense	メディアやユニットのパラメータ取得
$25	1	$5	Read Capacity	ユニットのブロック長やブロック数の取得
$28	1	$8	Read	拡張READ (ブロックアドレスや転送長の拡張)
$2A	1	$A	Write	拡張WRITE (〃)
$18	0	$18	Copy	論理ユニット間/同一ユニットでのコピー
$39	1	$19	Compare	〃 データ比較
$3A	1	$1A	Copy And Verify	〃 コピーとベリファイ

このうち，上から9つまでは，SCSIドライバがサポートを要求している必須コマンドです。X 68000でSCSIディスクを接続するときには，最低限，これらのコマンドがサポートされていなくてはなりません。

続く$28と$2Aの2つのコマンドは，6バイト長コマンドのReadコマンドとWriteコマンドを拡張し，より大きなブロック番号とブロック数の指定が行えるようにした拡張READ/WRITEコマンドです。X 68000では，ディスクの先頭ブロックに書き込まれるデバイスパラメータ中に拡張READ/WRITEコマンドが使用できるか否かを示すフラグがあります。

最後の3つ，Copy，Compare，Copy And Verifyコマンドは，とくに使用されることはないと思いますが，後の説明の中でこれらのコマンド名が出てくるため，一応オペレーションコードだけはあげておきます。

❻·3 SCSIの主要コマンドの内容

SCSIコマンドのうち，$00，$01，$03，$04，$08，$0Aの各コマンドは，SASIのところ

で説明したものと同様ですので省略し，ここでは，$12(INQUIRY)，$1A(MODE SENSE)，$25 (READ CAPACITY)，$28 (拡張 READ)，$2A (拡張 WRITE) の各コマンドについて説明していくことにします。

❻·❸1 INQUIRYコマンド(オペレーションコード$ 12)

INQUIRY コマンドのフォーマットを図 25 に示します。

●図……25 INQUIRY コマンド

転送順序	bit7	6	5	4	3	2	1	bit0	備考
0	'0'	'0'	'0'	'1'	'0'	'0'	'1'	'0'	オペレーションコード:$12
1	LUN (論理ユニット番号)			Reserved					
2	Reserved								
3	Reserved								
4	Allocation Length								イニシエータが用意しているバッファのバイト長
5	V		Reserved				Flag	Link	

このコマンドはターゲットと，その下に接続されているユニット（デバイス）がどのようなデバイスであるか，取り外し可能であるかなどといった，属性情報の読み出しを行います。このコマンドで得られるデータのフォーマットを図 26 に示します。

先頭バイトは，接続されているのが HDD のようなダイレクトアクセス(ランダムアクセス)デバイスであるか，シーケンシャルアクセスデバイスであるかなどのデバイスの種別を示すことにします。X 68000 では，現在，ダイレクトアクセスデバイスしかサポートされていませんが，将来は CD-ROM や DAT などのシーケンシャルアクセスデバイスのサポートも行われるようになるでしょう。

RMB ビットは，そのデバイスがリムーバブル(取り外し可能)であるか否かを示します。HD のように取り外し不可能な場合には RMB ビットは'0'，光磁気ディスクのようにリムーバブルなデバイス場合には'1'になります。

準拠規格は，そのデバイスが準拠している規格を判断するのに使用されます。下位 3 ビットが ANSI の規格，そのほかのビットが ISO や ECMA などで規定する SCSI 規格への準拠を示しますが，当然のことながら，ANSI の規格書では ANSI ビットの定義しかありません。一般的な SCSI 対応ハードディスクも，ISO や EMCS のビットはすべて'0'にしているようで

●図……26　INQUIRYデータ

転送順序	bit7	6	5	4	3	2	1	bit0	備　考
0	Peripheral Device Type								デバイス種別
1	RMB	Device-Type Qualifier							下位7bitは任意使用可(DIPスイッチの状態など)
2	ISO Version		ECMA Version			ANSI-Approved Version			準拠規格
3	(Reserved)								
4	Additional Length								追加データ長
5～n+4	Vendor Unique Parameter Bytes								追加データ

RMB:取り外し可能デバイスのとき'1'

Peripheral Device Type

値	内　容
$00	ダイレクトアクセスデバイス(HDD等)
$01	シーケンシャルアクセスデバイス(MT等)
$02	プリンタデバイス
$03	プロセッサデバイス
$04	WORM(追記型)デバイス
$05	読み出し専用ダイレクトアクセスデバイス
$06～$7E	(将来拡張用)
$7F	論理ユニットは存在しない
$80～$FF	各ベンダ(メーカ)で自由に使用可

ANSI-Approved Version

値	内　容
0	準拠規格なし
1	ANSI X3.131-1986準拠
2	ANSI X3T9.2/86-109(SCSI-2) 準拠
3～7	(将来拡張用)

す。

ANSIビットは'1'のとき，ANSI X 3.131-1986(これがもっとも一般的なSCSIの規格)に，'2'のときにANSI X3T9.2/86-109(SCSI-2)に準拠していることを示すことになっています。準拠規格がX 3.131-1986以前のものであるような場合には'0'を返すことになっています。

❻・❸ 2 MODE SENSEコマンド(オペレーションコード$1A)

メディアや論理ユニット，周辺デバイスパラメータなどを報告するコマンドです。コマンドのフォーマットは496ページの図27のようになっています。

このコマンドに対して，ターゲットは図28のようなフォーマットのデータを送ってきます。このうち，とくに必要性が高いのはWP (Write Protect＝書き込み禁止) ビットでしょう。'1'のとき，そのメディアが書き込み禁止であることを示します。

●図……27　MODE SENSE コマンド

転送順序	bit7	6	5	4	3	2	1	bit0	備　考
0	'0'	'0'	'0'	'1'	'1'	'0'	'1'	'0'	オペレーションコード：$1A
1	LUN (論理ユニット番号)			(Reserved)					
2	(Reserved) PC		(Reserved) (ページコード)						ANSI X3.131-1986ではReserved
3	(Reserved)								
4	Allocation Length								イニシエータが用意しているバッファのバイト数
5	V		(Reserved)				Flag	Link	

P C	内　容
'00'	カレント値
'01'	変更可能値
'10'	デフォルト値
'11'	セーブ値

ページコード	内　容
0	ページディスクリプタは転送しない
1	リード/ライトエラーリカバリパラメータ
2	ディスコネクト/リコネクトパラメータ
3	フォーマットパラメータ
4	ドライブパラメータ
7	ベリファイエラーリカバリパラメータ
8	キャッシングパラメータ
$21	アディショナルエラーリカバリパラメータ
$22	リコネクションタイミングパラメータ
$3F	全パラメータ

●図……28　MODE SENSE データ

転送順序	bit7	6	5	4	3	2	1	bit0	備　考	
0	Sense Data Length								センスデータ長(自分自身は含まない)	
1	Medium Type								メディアタイプ	
2	WP	(Reserved)							WP：Write Protect('1'のとき書き込み禁止)	
3	Block Descriptor Length								ブロックディスクリプタ長（8の倍数になる）	
0	Density Code								密度コード	ブロック ディスクリプタ (複数になることもある)
1	Number of Blocks (MSB)								ブロック数	
2	Number of Blocks									
3	Number of Blocks (LSB)									
4	(Reserved)								(将来拡張用)	
5	Block Length (MSB)								ブロック長	
6	Block Length									
7	Block Length (LSB)									
0～n	Vendor Unique Parameter Bytes								ベンダ(メーカ)ごとに自由に使用可	

Density Code

値	内　容
$00	デフォルト(単一の密度のみサポート)
$01	単密度フロッピーディスク
$02	倍密度フロッピーディスク
$03～$7F	(将来拡張用)
$80～$FF	ベンダ(メーカ)ごとに自由に使用可

メディアタイプは，おもにフロッピーディスクやMT (Magnetic Tape) を考えたパラメータであるため，HDでは$00が入るだけのようです。メディアタイプの内容を図29に示しておきます。

コマンド中のAllocation Lengthは，イニシエータが受け取りたいMODE SENSEデータのバイト数を指定します。ターゲットは，ここで指定されたバイト数以上のMODE SENSEデータは送信してきません。

また，コマンドの転送順序2のデータは，ANSI X 3.131-1986では予約領域となっているのですが，その後の標準化作業でPCとページコードというデータになったようです。残念ながら，私の手元には資料がないのですが，ディスクメーカの出しているマニュアルなどを見る

●図……29　メディアタイプ

値	メディアタイプ
$00	デフォルトメディア(currentry mounted medium type)
$01	片面フロッピーディスク(unspecified medium)
$02	両面　　〃　　(　　〃　　)

フロッピーディスクのメディアタイプ

値	サイズ	ビット密度 Bits/Radian	トラック密度 /mm (/inch)	面	参照規格
$05	8インチ	6　631	1.9 (48)	1	ANSI X3.73-1980
$06	〃	6　631	〃	2	EMCA 59
$09	〃	13　262	〃	1	なし
$0A	〃	13　262	〃	2	ANSI X3.121-1984
$0D	5.25インチ	3　979	〃	1	ANSI X3.82-1980
$12	〃	7　958	〃	2	ANSI X3.125-1985
$16	〃	7　958	3.8 (96)	2	ANSI X3.126-1986
$1A	〃	13　262	〃	2	ISO DIS8630-1985
$1E	3.5インチ	7　958	5.3(135)	2	ANSI X3.137

ダイレクトアクセスMT

値	幅 (mm)	トラック数	密度 ftpmm(ftpi)	参照規格
$40	6.3	12	394(10000)	ANSI X3B5/85-151
$44	6.3	24	394(10000)	〃

と，MODE SENSEデータのうち，Vendor Unique Parameter Bytesのところに，エラー発生時のリトライ回数や回復方法，欠陥ブロックの交替処理方法，シリンダ数，ヘッド数などの情報が入っています。PCとページコードで，これらのうち，どのパラメータを受け取りたいのかを指定しているわけです。これらの一部は，MODE SELECTコマンド(オペレーションコード　$15)で変更することも可能となっています。これらのパラメータの具体的な内容は非常に複雑であるわりには日常的に使用することはほとんどないので説明は省略します。一応，参考にしたドライブメーカ(富士通)のPCと，ページコードの値と，その内容を図27に併記しておきますので参考にしてください。

❻・❸ 3　READ CAPACITYコマンド(オペレーションコード $25)

コマンドフォーマットを図30に示します。このコマンドは，ドライブの1ブロックのバイト数と，ブロック数を報告させるものです。このコマンドに対する応答データのフォーマットを図31に示します。X 68000のSCSIドライバでは，ブロック長として256，512，1024バイトのいずれでもかまわないようにしており，ディスクの先頭のSCSIデバイスパラメータ領域にブロック長と総ブロック数を書き込んでいます。

●図……30　READ CAPACITYコマンド

転送順序	bit7	6	5	4	3	2	1	bit0	備　考
0	'0'	'0'	'1'	'0'	'0'	'1'	'0'	'1'	オペレーションコード：$25
1	LUN (論理ユニット番号)			(Reserved)				Rel Adr	
2	Logical Block Address (MSB)								論理ブロックアドレス (PMIビットが'0'のときはすべて0にすること)
3	Logical Block Address								
4	Logical Block Address								
5	Logical Block Address (LSB)								
6	(Reserved)								
7	(Reserved)								
8	V		(Reserved)					PMI	
9	V		(Reserved)				Flag	Link	

PMI (Partial Medium Indicator)
'0'：返されるブロックアドレスとブロック長データはユニットの最終ブロックの情報である。
　Logical Block Addressはすべて0にすること。
'1'：返されるブロックアドレスは，指定されたLogical Block Address以降で実質に使用できる最終ブロックのアドレスとなる。

●図……31　READ CAPACITYデータ

転送順序	bit7	6	5	4	3	2	1	bit0	備考
0	Logical Block Address(MSB)								論理ブロックアドレス
1	//								
2	//								
3	// (LSB)								
4	Block Length(MSB)								ブロック長(バイト単位)
5	//								
6	//								
7	// (LSB)								

❻・❸ 4 拡張READコマンド(オペレーションコード$28)

コマンドフォーマットを図32に示します。READコマンドを拡張して，より大きなブロック番号と転送ブロック数を指定できるようにしたものです。

転送順序1のRel Adrは，最後にアクセスしたブロック番号からの相対値であることを示すフラグで，相対値にする場合には'1'にします。

●図……32　拡張READコマンド

転送順序	bit7	6	5	4	3	2	1	bit0	備考
0	'0'	'0'	'1'	'0'	'1'	'0'	'0'	'0'	オペレーションコード:$28
1	LUN (論理ユニット番号)			(Reserved)				Rel Adr	
2	Logical Block Address (MSB)								読み始めるブロックアドレス
3	Logical Blok Address								
4	Logical Block Address								
5	Logical Block Address (LSB)								
6	(Reserved)								(将来拡張用)
7	Transfer Length (MSB)								読み出すブロック数
8	Transfer Length (LSB)								
9	V		(Reserved)				Flag	Link	

❻・❸ 5 拡張WRITEコマンド(オペレーションコード$2A)

コマンドフォーマットは図33のようになっています。拡張READコマンドと同様，より大きなブロック番号と，書き込みブロック数を指定できるようにしたものです。

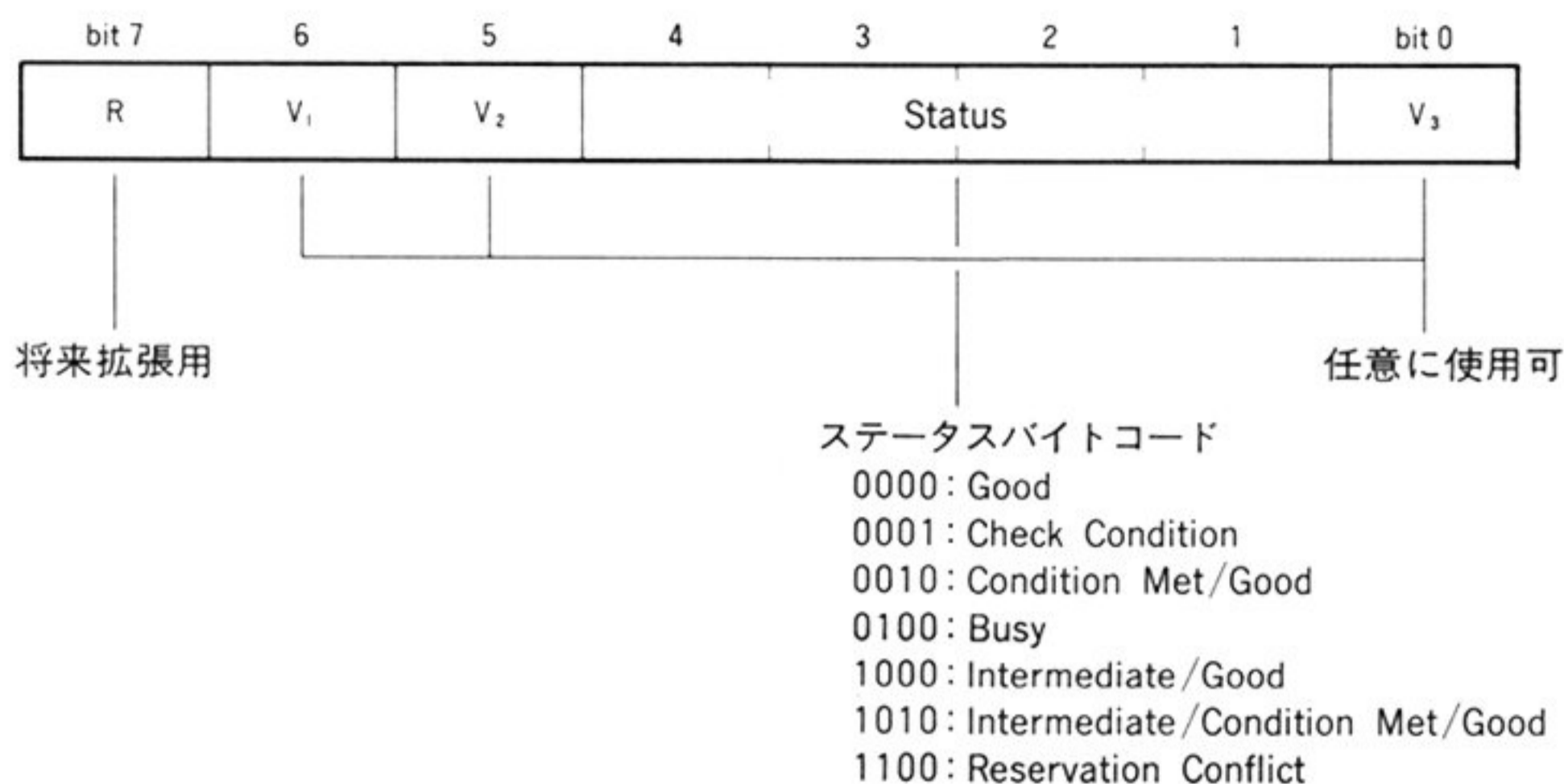
●図……33 拡張WRITEコマンド

転送順序	bit7	6	5	4	3	2	1	bit0	備考
0	'0'	'0'	'1'	'0'	'1'	'0'	'1'	'0'	オペレーションコード：$2A
1	LUN (論理ユニット番号)			(Reserved)				Rel Adr	
2	Logical Block Address (MSB)								書き込みを開始するブロックアドレス
3	Logical Block Address								
4	Logical Block Address								
5	Logical Block Address (LSB)								
6	(Reserved)								(将来拡張用)
7	Transfer Length (MSB)								書き込みを行うブロック数
8	Transfer Length (LSB)								
9	V		(Reserved)				Flag	Link	

●7 ステータスバイト

SASIでは，ステータスバイト(ステータスフェーズで渡されるデータ)は$00が正常という以外，特別な規定がなかったのですが，SCSIではよく使用されるものについて，コードの割り振りが決められました。501ページの図34にステータスバイトの内容を示します。

ビット1からビット4までがSCSIで規定されているステータスバイトコードです。これらの内容は，次のようになっています。

●図……34　ステータスバイトのフォーマット

Good

ターゲットは正常にコマンド処理を終了した

Check Condition

センスデータに反映させるエラー，例外，異常状態などが起こった。このステータスを受け取った場合，イニシエータは，REQUEST SENSE コマンドを使って，センスデータを受け取らなくてはならない

Condition Met

サーチコマンドでデータが見つかったときに返される。論理ブロックアドレスは，センスデータ（REQUEST SENSE コマンドを発行したときの返答データ）でわかる

Busy

ターゲットはビジーである。イニシエータは，しばらく待ってから，コマンドを再発行することで回復できる（かもしれない）

Intermediate

コマンドの連続実行機能(コマンドのコントロールバイトの Link フラグを立てる)を使用したときのコマンド処理終了メッセージとして返答される

Reservation Conflict

指定したドライブは，他のイニシエータからリザーブされている状態であり，解除されるまで使用不可能である(通常，X 68000 の SCSI システムではイニシエータとなるのは X 68000 だけなので，これが返ってくることはない)

Check Condition が返ってきたときには，必ずその直後に REQUEST SENSE コマンドを発行する必要があります。大方の SCSI デバイスは受け取るまで，REQUEST SENSE 以

外のコマンドは実行されなくなってしまうようです。

また，ドライブがリセットされたり，電源をON/OFFされると，最初のコマンドに対する応答は必ずCheck Conditionとなります。

●8 センスデータ

REQUEST SENSEコマンドに対する応答として返されるセンスデータは，SASI相当の4バイトデータでは少々情報不足であるということから，SCSIではあらたに拡張型センスデータフォーマットが定められました。

拡張型センスデータのフォーマットを図35に示します。先頭バイトの下位7ビットが$70であるとき，拡張型センスデータであることを示します。

●図……35 拡張型センスデータのフォーマット

転送順序	bit7	6	5	4	3	2	1	bit0	備考
0	V	'1'	'1'	'1'	'0'	'0'	'0'	'0'	拡張センスデータであることを示す
1	セグメント番号								Copy, Compare, Copy & Verifyの異常終了時，実行中のセグメント番号を示す
2	FM	EOM	ILI	(R)	センスキー				
3	インフォメーションバイト(上位)								インフォメーションバイト
4	〃								
5	〃								
6	インフォメーションバイト(下位)								
7	追加センスデータ長								
8～n+7	追加センスデータ								

V (Valid):インフォメーションバイトの内容が有効なとき'1'
FM(File Mark):シーケンシャルアクセスデバイスのとき，ファイルマークが検出されたことを示す
EOM (End of Medium): 〃 媒体の終了が 〃
ILI(Incorrect Length Indicator):データブロック長の不一致が検出されたことを示す
(R):リザーブ 将来拡張用

センスデータの転送順序2の下位4ビットはセンスキーと呼ばれるデータで，これによって，そのセンスデータがどのようなものであるのかを示します。センスキーの値と，その内容との対応は図36のようになっています。また，図37と図38に，それぞれREADコマンドとWRITEコマンドで発生する代表的なエラーと，それに対応するセンスキーを示しますので参考にしてください。

●図……36　センスキーと内容

センスキー値	名　称	内　　容
$0	No Sense	特有のセンスキーはない
$1	Recovered Error	最後に与えられたコマンドがリカバリ動作により正常終了した
$2	Not Ready	指定されたユニットはアクセス可能な状態ではない
$3	Medium Error	媒体の欠陥や記録されたデータの異常による回復不可能なエラー
$4	Hardware Error	回復不可能なハードウェアエラー (コントローラ，デバイスの故障など)
$5	Illegal Request	コマンドやパラメータに不正な値が検出された
$6	Unit Attention	メディアの入れ替えやユニットのリセットが行われた
$7	Data Protect	プロテクトされた領域にリード/ライトしようとした
$8	Blank Check	読み出し中にブランク領域になった*1,*2 書き込み中にブランクでない領域になった*1
$9	Vendor Unique	ベンダ(メーカ)ごとに自由に使用可
$A	Copy Aborted	Copy, Compare, Copy & Verifyコマンドがデバイス異常により中止した
$B	Aborted Command	ターゲットはコマンドの実行を異常終了した
$C	Equal	Searchコマンドで一致を検出した
$D	Volume Overflow	データがバッファに残っているのに，デバイスは最終ブロックに達してしまった
$E	Miscompare	ソースデータとメディアから読み出したデータが一致しない
$F	(Reserved)	(将来拡張用)

＊1:追記型デバイスのとき
＊2:シーケンシャルアクセスデバイスのとき(MTなど)

●図……37　READコマンドに対するセンスキー値

状　況	センスデータ中のセンスキー値
無効なブロックアドレスを指定した	Illegal Request（最初に異常になったブロックアドレスも返される）
ターゲットがリセットされたり，メディアの交換が行われた	Unit Attention
回復不可能なリードエラー	Medium Error
回復可能なリードエラー	Recovered Error
オーバーラン/リトライで救済できるエラー	Aborted Command

●図……38　WRITEコマンドに対するセンスキー値

状　況	センスデータ中のセンスキー
無効なブロックアドレスを指定した	Illegal Request（最初に異常となったブロックアドレスが返される）
ターゲットがリセットされたり，メディアの交換が行われた	Unit Attention
オーバーラン/リトライで救済できるエラー	Aborted Command

9 メッセージデータ

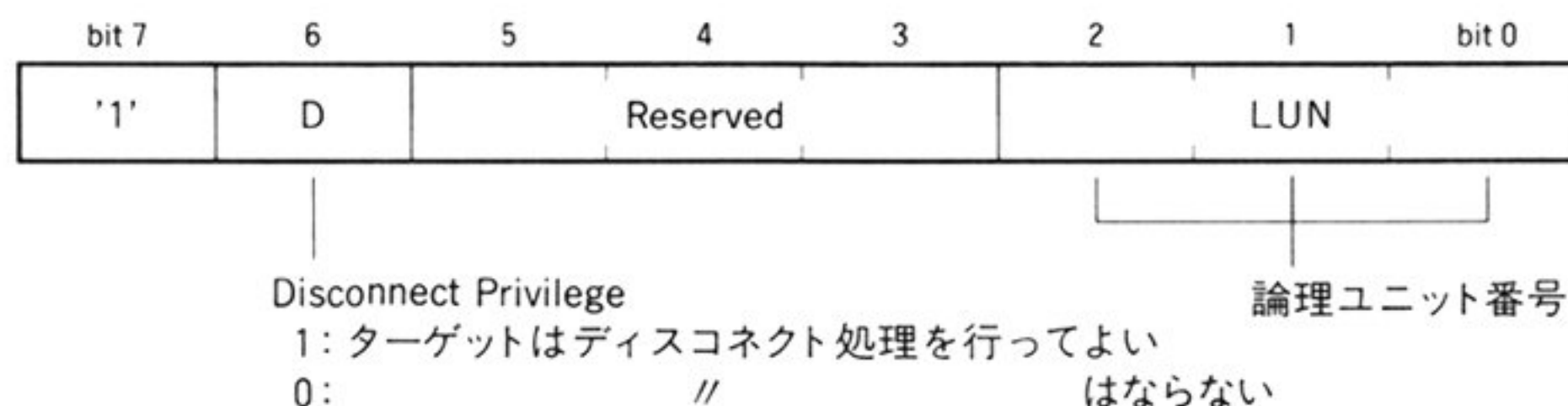
SASIのメッセージは，たんにコマンド処理の最後に送られるデータという以上の役目はありませんでしたが，SCSIではメッセージの意味が拡張され，それにともなって主要なメッセージについてはコードの規定が行われました。図39にSCSIで規定されているメッセージデータの一覧を示します。

●図……39　メッセージデータ

コード	必須(M) オプション(O)	名　称	I/O	備　　考
$00	M	Command Complete	I	コマンド実行完了
$01	O	Extended Message	I/O	拡張メッセージの1バイト目
$02	O	Save Data Pointer	I	カレントデータポインタの退避要求
$03	O	Restore Pointers	I	退避していたデータポインタの復帰
$04	O	Disconnect	I	ターゲットからのバス結合中断通知
$05	O	Initiator Detected Error	O	イニシエータはエラーを検出した
$06	O	Abort	O	ターゲットの入出力動作をクリアする
$07	O	Message Reject	I/O	受け取ったメッセージはサポートされていない
$08	O	No Operation	O	なんら有効なメッセージを保持していない
$09	O	Message Parity Error	O	メッセージ受信中にパリティエラーを検出した
$0A	O	Linked Command Complete	I	リンク付きでフラグ='0'のコマンドの処理が正常終了した
$0B	O	Linked Command Complete(with Flag)	I	〃 フラグ ='1' 〃
$0C	O	Bus Device Reset	O	バス上で動作中/保留中のすべての入出力動作をクリア
$0D～$7F	―	(Reserved Codes)		（将来拡張用）
$80～$FF	O	Identify	I/O	イニシエータとターゲット間の入出力パスの設定

9·1 IDENTIFYメッセージ

IDENTIFY メッセージでは，図 40 のようにビットの割り振りが行われています。セレクションフェーズ直後のメッセージアウトフェーズで，ターゲットがディスコネクト処理を行ってよいか否かの選択を行うのに使用されます。

●図……40　IDENTIFY メッセージ

9·2 拡張メッセージ

コード$01 の EXTENDED MESSAGE は，複数バイトにわたる拡張メッセージであることを示すコードです。拡張メッセージのフォーマットは図 41 のようになっています。

●図……41　拡張メッセージのフォーマット

転送順序	値	備　　考
0	$01	拡張メッセージであることを示す
1	N	拡張メッセージ長
2		拡張メッセージコード
3～N＋1		拡張メッセージアーギュメント

先頭の$01 に続くデータで，拡張メッセージのメッセージコード以降のバイト数を示します。転送順序 2 のデータがメッセージの種別を識別するための拡張メッセージコード，それ以降は拡張メッセージに付随するアーギュメントとなっています。拡張メッセージコードは 506 ページの図 42 のように割り振られています。

●図……42 拡張メッセージコード

コード	名 称	I/O	備 考
$00	Modify Data Pointer	I	カレントデータポインタの値を増減する
$01	Synchronous Data Transfer Request	I/O	同期転送用のパラメータ定義
$02	Extended Identify	I/O	IdentifyメッセージのLUNを拡張する
$03～$7F	(Reserved)	—	(将来拡張用)
$80～$FF	(Vendor Unique)	—	各デバイスごとに自由に定義可

9·2 1 MODIFY DATA POINTERメッセージ

メッセージのフォーマットは図43のようになっています。ターゲットからイニシエータに対して，現在のポインタの値をアーギュメントで示される分だけ増減します。アーギュメントは2の補数表現です。

ポインタには，コマンドポインタ，データポインタ，ステータスポインタの3種類が想定されており，コマンドポインタはコマンド列の先頭，データポインタやステータスポインタはデータやステータスの格納位置を示します。

●図……43 MODIFY DATA POINTER メッセージ

転送順序	データ	備 考
0	$01	拡張メッセージであることを示す
1	$05	拡張メッセージ長
2	$00	Modify Data Pointerメッセージ
3	データ上位	データポインタの移動量 符号付き2進数 (2の補数表記)
4		
5		
6	データ下位	

9·2 2 SYNCHRONOUS DATA TRANSFER (同期データ転送要求)メッセージ

SCSIでは，SASIと同様のREQ-ACKハンドシェークによる転送のほか，ACKを待たずにREQ信号を連続して変化させることでデータを先行して送ってしまう同期転送機能の規定が行われました(この機能はオプションです)。これによって，データの転送速度を大幅に向上

させることが可能となります。残念ながら，X 68000 に使用されている SCSI コントローラ，MB89352 は同期転送モードをサポートしていませんので，このメッセージは使用できません。

データ転送の際にこのモードを指定するのが同期データ転送要求メッセージです。メッセージのフォーマットを図 44 に示します。転送順序 3 で転送の周期を指定します。転送順序 4 の REQ/ACK オフセットというのは，先行して送ることができるデータの数を示すものです。たとえば，この値が 4 であるなら，ACK が返ってこなくても，4 回（4 バイト）のデータ転送が行われることになります。

●図……44　同期データ転送要求メッセージ

転送順序	データ	備　考
0	$01	拡張メッセージであることを示す
1	$02	拡張メッセージ長
2	$02	Extended Identify メッセージ
3	X	サブ論理ユニット番号

❾・❷ 3　EXTENDED IDENTIFYメッセージ

通常，コマンドで指定できる論理ユニット番号は 0 から 7 までであるため，1 つのターゲットの下には論理ユニットを 8 台までつなぐことができるようになっています。これをさらに拡張するのが，このメッセージです。メッセージのフォーマットは図 45 のようになっています。このメッセージで与えられた 8 ビットのサブ論理ユニット番号とコマンドの中にある 3 ビットの論理ユニット番号とを組み合わせて最大 2048 台までの論理ユニットを指定することができるようになります。

●図……45　EXTENDED IDENTIFY メッセージ

転送順序	データ	備　考
0	$01	拡張メッセージであることを示す
1	$03	拡張メッセージ長
2	$01	Synchronous Data Transfer Request メッセージ
3	m	転送周期（4×m(ns)）
4	x	REQ/ACK オフセット

Human68KのSCSIドライバは論理ユニット番号を使用していませんので，このメッセージも使用されることはありません。

●10 サンプルプログラム

SCSIディスクから指定したブロックを読み出すサンプルプログラムを作成してみました。第1引き数でアクセスするブロック番号，第2引き数でアクセスするSCSIディスクのID番号を指定します。引き数が省略された場合には，それぞれ0として扱われます。

なお，このサンプルでは，ブロックサイズは512バイト，SCSIインタフェースはCZ-6BS1であるものとしています。ブロックサイズが512バイト以外である場合にはBUFSIZEの値を，SCSI内蔵タイプのときはspcの値を適宜変更してください。

●リスト……1　SCSIディスクからの指定ブロック読み出し

```
/*
 * ＳＣＳＩハードディスクアクセステスト
 *
 * XC ではvolatile がサポートされていないため、
 * 次の一行を入れてvolatileを無効にしてください
 * #define  volatile
 */

#include <doslib.h>

struct  DMAREG {
    unsigned char   csr;
    unsigned char   cer;
    unsigned short  spare1;
    unsigned char   dcr;
    unsigned char   ocr;
    unsigned char   scr;
    unsigned char   ccr;
    unsigned short  spare2;
    unsigned short  mtc;
    unsigned char   *mar;
    unsigned long   spare3;
```

```
    unsigned char   *dar;
    unsigned short  spare4;
    unsigned short  btc;
    unsigned char   *bar;
    unsigned long   spare5;
    unsigned char   spare6;
    unsigned char   niv;
    unsigned char   spare7;
    unsigned char   eiv;
    unsigned char   spare8;
    unsigned char   mfc;
    unsigned short  spare9;
    unsigned char   spare10;
    unsigned char   cpr;
    unsigned short  spare11;
    unsigned char   spare12;
    unsigned char   dfc;
    unsigned long   spare13;
    unsigned short  spare14;
    unsigned char   spare15;
    unsigned char   bfc;
    unsigned long   spare16;
    unsigned char   spare17;
    unsigned char   gcr;
} ;
volatile struct DMAREG  *dma;

struct  SPCREG {
    unsigned char   DUMMY0;
    unsigned char   bdid;
    unsigned char   DUMMY1;
    unsigned char   sctl;
    unsigned char   DUMMY2;
    unsigned char   scmd;
    unsigned short  DUMMY4;
    unsigned char   DUMMY3;
    unsigned char   ints;
    unsigned char   DUMMY5;
    unsigned char   psns;
    unsigned char   DUMMY6;
    unsigned char   ssts;
    unsigned char   DUMMY7;
    unsigned char   serr;
```

```
    unsigned char   DUMMY8;
    unsigned char   pctl;
    unsigned char   DUMMY9;
    unsigned char   mbc;
    unsigned char   DUMMY10;
    unsigned char   dreg;
    unsigned char   DUMMY11;
    unsigned char   temp;
    unsigned char   DUMMY12;
    unsigned char   tch;
    unsigned char   DUMMY13;
    unsigned char   tcm;
    unsigned char   DUMMY14;
    unsigned char   tcl;
};
volatile struct SPCREG *spc;

#define BUFSIZE 0x200
unsigned char   diskbuf[BUFSIZE];

#define PSNS_REQ     0x80
#define PSNS_ACK     0x40
#define PSNS_BUSFREE     0x00
#define PSNS_STATUS 0x0b
#define PSNS_MESSAGE     0x0f
#define PSNS_COMMAND     0x0a

#define PCTL_DATA_IN     0x1
#define PCTL_COMMAND     0x2
#define PCTL_STATUS 0x3
#define PCTL_MESSAGE     0x7

#define SCMD_SELECT 0x20
#define SCMD_SET_ACK     0xe4
#define SCMD_RESET_ACK  0xc4
#define SCMD_TRANSFER   0x80

#define INTS_DISCONNECT 0x20
#define INTS_COMPLETE   0x10
#define SSTS_DREG_EMPTY 0x01

void main();
void scsi_busfree();
```

```
void scsi_ints_wait();
void scsi_phase_wait();
void scsi_select();
void scsi_send_command();
void scsi_send_a_byte();
void scsi_data_transfer();
void scsi_buffer_wait();
unsigned int scsi_get_status();
unsigned int scsi_get_message();
unsigned int scsi_get_a_byte();
void dma_setup();
void dma_start();
void dma_stop();
void wait_complete();
void clear_flag();

void main(argc,argv)
    int argc;
    char    *argv[];
{
    unsigned int    i,j,id,blk_no,blk_h,blk_m,blk_l;
    unsigned char   c;
    if (argc >= 2)
        blk_no = atoi(argv[1]);
    else    blk_no = 0;
    if (argc >= 3)
        id = atoi(argv[2]);
    else    id = 0;
    SUPER(0);
    spc = (struct SPCREG *)0xea0000;
    dma = (struct DMAREG *)0xe84040;
    spc->bdid = 0x7;
    spc->sctl = 0x10;
    blk_l = blk_no & 0xff;
    blk_m = (blk_no >> 8) & 0xff;
    blk_h = (blk_no >> 16) & 0xff;
    printf("Block# = %d(%06X)[%02X:%02X:%02X] Drive = %d¥n",
                blk_no, blk_no, blk_h, blk_m, blk_l, id);
    printf("Bus Free¥n");
    scsi_busfree();
    printf("Select¥n");
    scsi_select(id);
    printf("Command¥n");
```

```
    scsi_send_command(8,blk_h,blk_m,blk_l,1,0);
    printf("Data In¥n");
    scsi_data_transfer();
    printf("Status = %02X¥n",scsi_get_status());
    printf("Message= %02X¥n",scsi_get_message());
    for (i=0; i<BUFSIZE; i+=0x10) {
        for (j=0; j<0x10; j++)
            printf("%02X ",diskbuf[i+j]);
        for (j=0; j<0x10; j++) {
            c = diskbuf[i+j];
            if ((c < 0x20) ||(c >= 0xe0)||((c >= 0x80) && (c < 0xa0)))
                printf(".");
            else    printf("%c",diskbuf[i+j]);
        }
        printf("¥n");
    }
}

void scsi_busfree()
{
    scsi_phase_wait(PSNS_BUSFREE);
    if (spc->ints & INTS_DISCONNECT)
        spc->ints = INTS_DISCONNECT;
}

void scsi_ints_wait(dat)
    unsigned int    dat;
{
    while(!(spc->ints & dat))
        ;
    spc->ints = dat;
}

void scsi_phase_wait(phase)
    unsigned char   phase;
{
    while(spc->psns != phase)
        ;
}

void scsi_select(id)
    unsigned int    id;
{
    spc->temp = (1 << id) | (spc->bdid);
```

```
    spc->tch  = 0;
    spc->tcm  = 0;
    spc->tcl  = 3;
    spc->pctl = 0;
    spc->scmd = SCMD_SELECT;
    scsi_ints_wait(INTS_COMPLETE);
}

void scsi_send_command(p1,p2,p3,p4,p5,p6)
    unsigned int    p1,p2,p3,p4,p5,p6;
{
    unsigned char   param[6];
    param[0] = p1;
    param[1] = p2;
    param[2] = p3;
    param[3] = p4;
    param[4] = p5;
    param[5] = p6;
    clear_flag();
    dma_setup(0,param, &(spc->dreg), 6);
    spc->tch = 0;
    spc->tcm = 0;
    spc->tcl = 6;
    spc->pctl = PCTL_COMMAND;
    spc->scmd = SCMD_TRANSFER;
    scsi_phase_wait(PSNS_COMMAND | PSNS_REQ);
    dma_start();
    wait_complete();
    scsi_ints_wait(INTS_COMPLETE);
}

void scsi_data_transfer()
{
    unsigned int    i;
    clear_flag();
    dma_setup(1, diskbuf, &(spc->dreg), BUFSIZE);
    spc->tch = (BUFSIZE >> 16) & 0xff;
    spc->tcm = (BUFSIZE >> 8) & 0xff;
    spc->tcl = BUFSIZE & 0xff;
    spc->pctl = PCTL_DATA_IN;
    spc->scmd = SCMD_TRANSFER;
    scsi_buffer_wait();
    dma_start();
```

```
    wait_complete();
    scsi_ints_wait(INTS_COMPLETE);
}

void scsi_buffer_wait()
{
    while(spc->ssts & SSTS_DREG_EMPTY)
        ;
}

unsigned int scsi_get_status()
{
    spc->pctl = PCTL_STATUS;
    scsi_phase_wait(PSNS_STATUS | PSNS_REQ);
    return(scsi_get_a_byte());
}

unsigned int scsi_get_message()
{
    spc->pctl = PCTL_MESSAGE;
    scsi_phase_wait(PSNS_MESSAGE | PSNS_REQ);

    return(scsi_get_a_byte());
}

unsigned int scsi_get_a_byte()
{
    unsigned int    dat;
    while (!(spc->psns & PSNS_REQ))
        ;
    dat = spc->temp;
    spc->scmd = SCMD_SET_ACK;
    while (spc->psns & PSNS_REQ)
        ;
    spc->scmd = SCMD_RESET_ACK;
    while(spc->psns & PSNS_ACK)
        ;
    return(dat);
}

void dma_setup(dir,ma,da,len)
    unsigned int    dir,len;
    unsigned char   *ma,*da;
{
```

```
    dma->dcr = 0x80;
    dma->ocr = 0x31 | ((dir & 0x1) << 7);
    dma->scr = 0x04;
    dma->ccr = 0x00;
    dma->cpr = 0x08;
    dma->mfc = 0x05;
    dma->dfc = 0x05;

    dma->mtc = len;
    dma->mar = ma;
    dma->dar = da;
}

void dma_start()
{
    dma->ccr |= 0x80;
}

void wait_complete()
{
    while(!(dma->csr & 0x90))
        ;
}

void clear_flag()
{
    dma->csr = 0xff;
}
```

システムポート

システムポートには，ディスプレイやキーボード，電源OFFコントロールなど，こまごまとした周辺制御用の信号が集められています。ここでは，各システムポートの内容と，その操作方法について説明します。

1 システムポートのアドレス配置

システムポートは，ディスプレイのコントラストの設定や電源ON/OFF制御などのサポートを行うもので，6つのレジスタから構成されています。それぞれのアドレス配置は518ページの図1のようになっています。

1·1 システムポート＃1

コンピュータ画面のコントラストの調整を行います。下位4ビットで明るさの度合が決まり，$Fがもっとも明るく，$0がもっとも暗くなります。Human 68 Kは，通常は$Eで使用しており，電源OFFなどのときはこれを使って画面を暗くしてから落ちるようにしています。

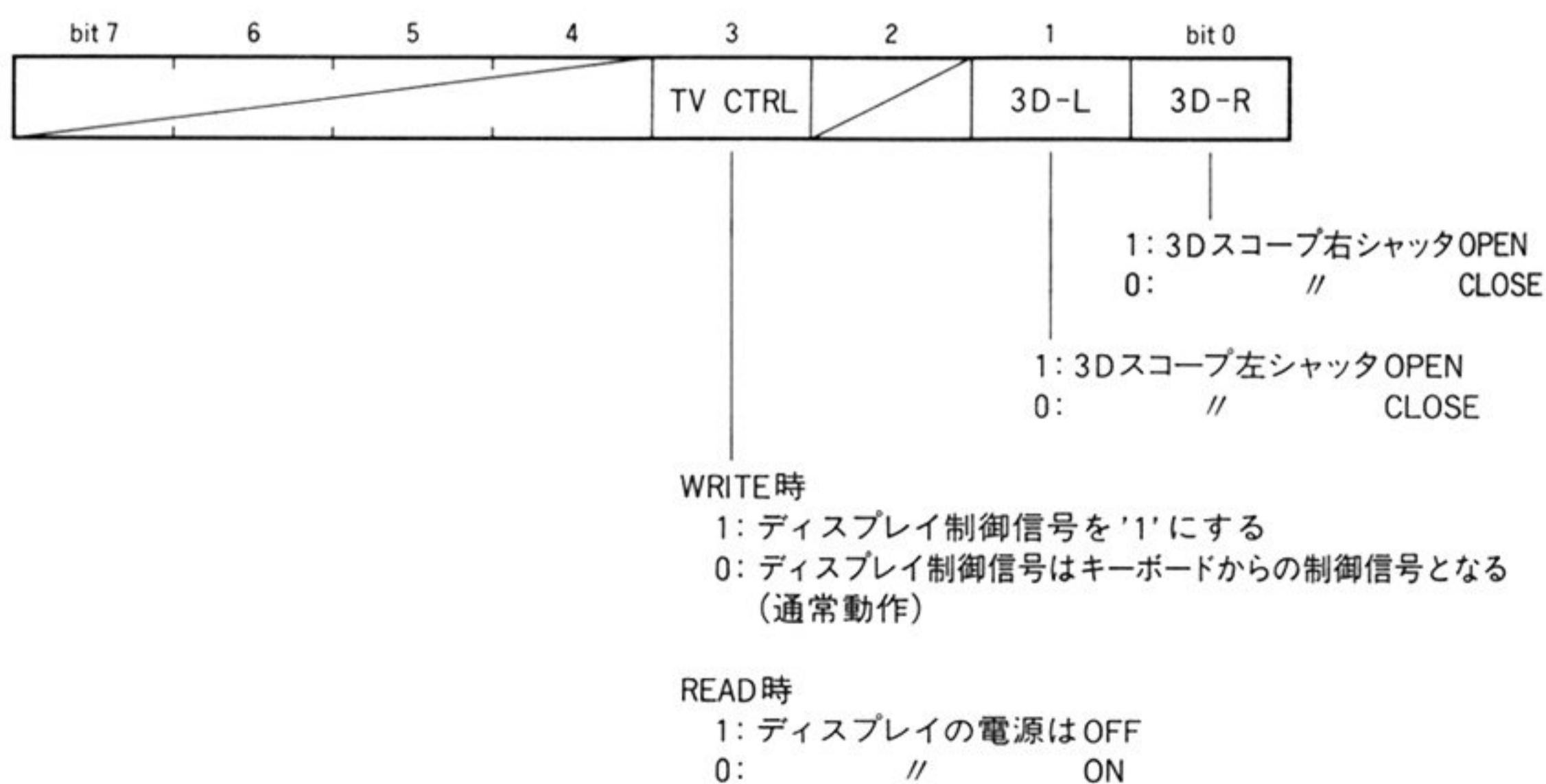
●図……1　システムポート

レジスタ#	アドレス	bit 7	6	5	4	3	2	1	bit 0	備　考
1	$E8E001					CONTRAST				コンピュータ画面コントラスト
2	$E8E003					TV CTRL		3D-L	3DL-R	ディスプレイ/3Dスコープ制御
3	$E8E005				カラーイメージユニット制御					
4	$E8E007					KEY CTRL	NMI RESET	HRL		キーボード/NMI
5	$E8E00D	SRAM Write Enable Control								SRAM書き込み制御
6	$E8E00F					Power OFF Control				本体電源OFF制御

❶·2　システムポート♯2

ビット配置を図2に示します。下位2ビットはオプションの3Dスコープの制御に用いられるものです。ビット0が右目，ビット1が左目のシャッターに対応しており，それぞれ'1'になっていると，シャッターがOPENし，画面が見えるようになります。

ビット3は，書き込み時はディスプレイ制御信号，リード時はディスプレイの電源のON/OFFステータスとして動作します。このビットの詳細については，キーボードの説明のページを参照してください。

●図……2　システムポート#2($E8E003)

❶·3 システムポート＃3

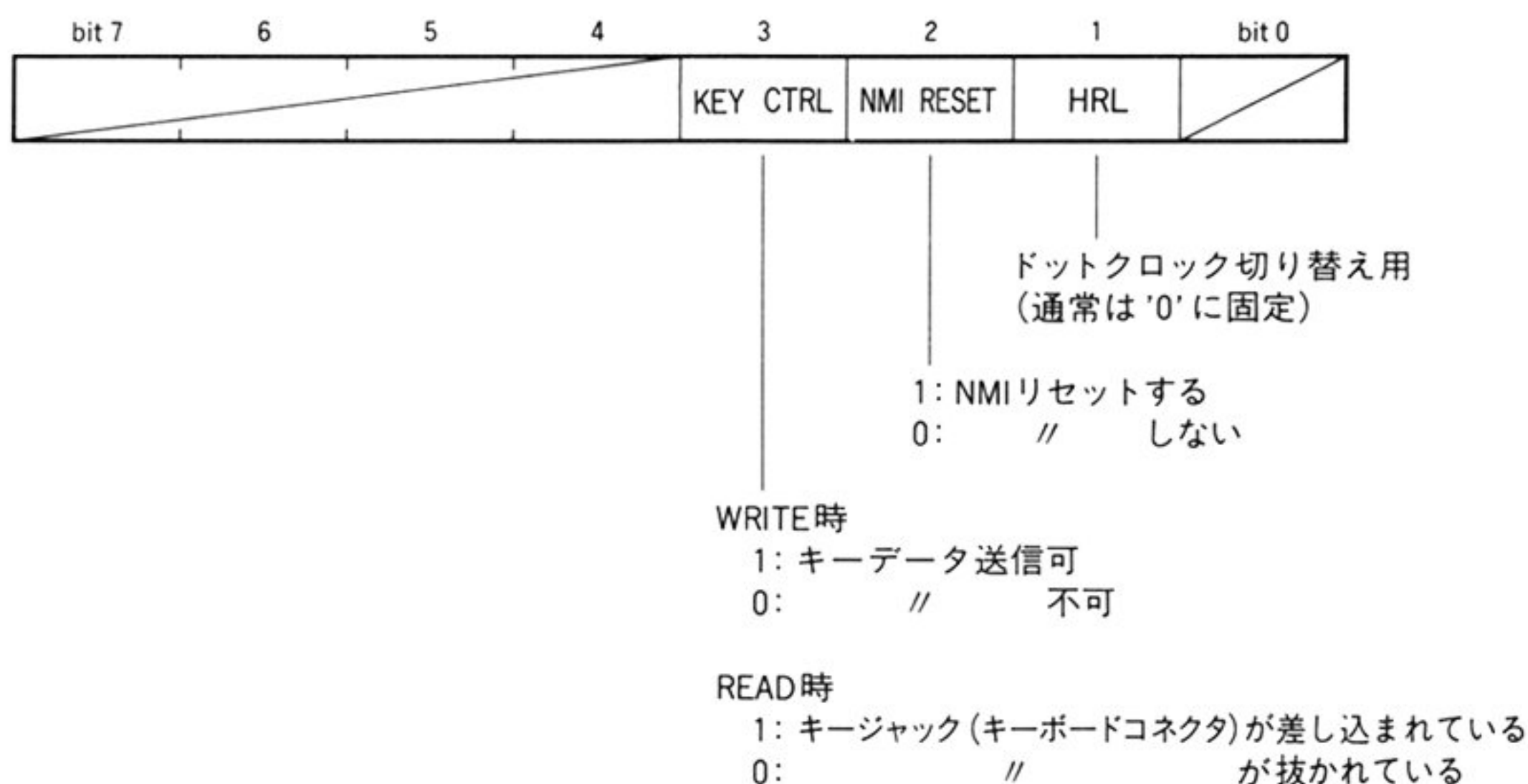
システムポート＃3はオプションのカラーイメージユニットの制御に使用されるものです。ここに書き込んだ値は，そのままIMAGE IN端子の17〜21番ピン(17がビット4，21がビット0に対応）に出力されます。

❶·4 システムポート＃4

ビット配置は図3のようになっています。ビット1は，ドットクロックの切り替え時に使用するものですが，通常は'0'のままにしておきます。

ビット2は，NMIが発生したとき，NMIの処理が終了した時点で'1'を書き込むビットです。一度NMIが発生すると，このビットに'1'を書き込まないかぎり，次のNMIが発生しなくなります。

ビット3は，キーボードのCPUの制御やキーボードコネクタが差し込まれているか否かのチェックを行うものです。このビットの詳細はキーボードの説明を参照してください。

●図……3　システムポート＃4($E8E007)

❶·5 システムポート＃5

このポートはSRAMの書き込み許可/禁止を制御するものです。SRAMにはメモリ容量などのシステム情報や起動デバイス，キーボードの文字選択などの情報が書き込まれるようになっているため，プログラムミスなどがあっても，容易に書き換わらないようにしておく必要があります。

このようなことから，SRAMへの書き込み保護のために設けられたのが，このポートです。このポートに$31を書き込むとSRAMへの書き込みが許可に，それ以外のデータを書くと書き込み禁止になります。

❶·6 システムポート＃6

本体の電源OFFを行うものです。正面の電源スイッチがOFFになっているときに，このポートに$00，$0F，$0Fと順に書き込むことで，本体の電源をOFFにすることができます。不用意に電源が落ちるのを防ぐため，この順序で書き込まなければ働かないようになっています。

本体正面の電源スイッチがOFFになると，MFPのGPIP 2の割り込みが発生しますので，通常はこの割り込み処理の中でこのポートを操作して電源を落とします。実験するときは，GPIP 2の割り込みを禁止しないと，電源スイッチをOFFにしたとたん，Human 68 Kの電源OFF処理が働いてしまいますので注意してください。

厄介な仕事を引き受けてしまったものだといまさらながら思っています（私のつたない説明を読まされるほうがもっとたいへんだともいえるかもしれませんが）。

X 68000のハードウェアに触れた本は，1987年に X 68000が発売されて1年くらいの間は数冊あったようなのですが，これらはすでに絶版になってしまったらしく，いまでは書店に行っても，まったく見当たりません。X 68000ユーザの数も相当いるのだから，また新しい解説書がどこかから必ず出てくるだろうとじっと待っていたのですが，いつまでたっても出てくる気配がありません。そんなとき，「X 68000のハードウェア解説書を書いてみないか」という話がきてしまいました。完成品の無線機が買えなくてキットを組み立て，TK-80が買えなくてICを1つずつ買い集めてユニバーサル基板で自作してきた私には，「なければ自分でつくらなくてはならない」というのは宿命というものだったのでしょうか。

自分でやる以上は，これまでさんざん不愉快な思いをしてきたLSIマニュアルへの依存を断ち切ろうと決めました。8ビット時代からいままで，いろいろなハードウェアに触れてきましたが，比較的原始的なLSIばかりで構成され，しかもユーザも相当数いるはずの86系パソコンのハードウェア解説書ですら，ていねいに説明しているのはCRTまわりだけで，ほかはポートアドレスやレジスタのビット配置だけを載せて，「詳細はそれぞれのLSIのマニュアルを参照してください」ですませてしまっているのが大多数です。

これらの筆者の方々はおそらく大学の研究室やメーカの研究・開発部門など，マニュアル類は電話ひとつで手に入るような立場におられるのでしょう。実際にこのような職業上の特権がない者がLSIのマニュアルなどを手に入れようとすれば，秋葉原の部品屋でコピーしてもらったり，CQ出版社が出しているものを注文するよりありません（1冊3000円以上するのが普通，どうかすると1万円以上とられることもあります）。これらにしても，まだ見つかれば好運なほうで，まったく手に入らないことも珍しくありません（筆者も，今回の執筆中，あるLSIのマニュアルがどうしても見つからず，とうとう展示会のときにブースの方に泣きついて名刺と引き換えでもらうという手段をとってしまいました）。

また，一度でも読んだことがある方でしたら，よくご存じでしょうが，LSIのマニュアルというのはお世辞にも読みやすいといえるようなものではありません。何度読み返しても，いった

い何がいいたいのかよくわからず，結局，プログラムをつくって動作チェックをしているうちにようやく意味がわかるといったこともよくありました。

LSIのマニュアルの内容をそのまま全部書き直すようなことはとてもできませんが，とにかくLSIのマニュアルがなくても，なんとかなる程度には説明しておくことにしようという方針だけは決めました。しかし，この方針が後でどれだけ自分を苦しめることになるか，そのときは想像もできませんでした。「なんでこんなにややこしいんだ！」と，何度頭を抱え込んだことか知れません。

これまで86系CPUのパソコンは仕事がらみもあってかなり扱っていたので，わりと軽く考えていたのが大間違いでした。86系のパソコンの代表であるIBM PCにしてもPC-9801にしても，内部のI/OデバイスはCPUの能力からすると信じられないほど低レベルなものばかりです。64K境界をまたいだ転送すらできず，CPUよりも低速なDMA，バンク切り替えだらけでようやく16色しか出せないグラフィックVRAM，I/Oポートにスピーカをつないだだけの音出力など，8ビットパソコンのCPUだけを載せ替えたような，そのハードウェア構成に知らず知らずのうちに慣らされ，パソコンとはそういうものだという意識を植え付けられてしまっていたのかもしれません。

それらに比べると，X68000ではCRTC，スプライトコントローラ，ディスプレイコントローラ，DMA，SCC，SPC，OPM，ADPCM……，シャープが独自開発したLSIもさることながら，その他のLSIにしても，86系CPUの一般的なパソコンのものとは比べものにならないほど，高度なものばかりです。RS-232Cにしても，通常は非同期無手順でしか使われないにもかかわらず，あえてデータの変復調機能まであるZ8530 SCCを採用しています。サンプリング音源もいくつものメーカがさまざまな方式のものを発表していますが，沖電気のADPCMチップはサンプリングレートのわりにはかなり音がよいほうであるという評価を受けているという話でした（おかげでADPCMのアルゴリズムは企業秘密であるとして教えてもらえなかったというオチまでついています）し，FM音源LSIも，価格を聞いてみると，ヤマハが出している各種のFM音源LSIの中でももっとも高価なものを使っているのです。

本書を執筆するために集めた資料や情報のメモ書きで筆者のこたつの上はまさに紙の山と化してしまいました（先日，首都圏を襲った震度5の地震でこの山もついに崩れ落ちました）。

さらに，これらの各LSIの機能の多さに加え，LSIのマニュアルの読みづらさ，間違いの多さにもほとほと閉口させられました。たとえば，「浮動小数点演算プロセッサ68881はI/Oとして使えます」といった説明がありながら，本文では68020に直結したときの説明ばかりで，X68000のようにI/Oとして使った場合の具体的な例などはほとんど説明されていません。また，セカンドソースメーカの日本語マニュアルでは，"SUBSTRUCT(除算)"などと堂々と書いている始末です(加減乗除という言葉を知らなかったのだろうか)。セカンドベンダとはいえ，仮りにも，これがLSIメーカの正式のマニュアルなのですから，ほかはもう推して知るべしで

しょう（結局，英文マニュアルを入手して辞書を片手に読むハメになってしまいました。英語ができなかったから理系に進んだようなものなのに……）。

X 68000 本体については，以前出版されていた解説書も参考にしつつ，極力動作チェックをしながら進めていたのですが，こちらもところどころ動きが変なところがありました。どうやら，初代機が発売されるまでの間に仕様が変更されたらしいのです。このようなところを見つけるたびに，チェック用のプログラムをつくりなおしたり，シンクロを持ち出して信号を調べたりと，大騒ぎになっていました。

当初，遅くとも年内には脱稿する予定だったのですが，調べても，解決しても，次から次へと現れる難関と格闘（物理的な難関――座卓でキーボードを叩いていると，いつの間にか忍び寄っている娘（昨年の6月に生まれた）の攻撃を足で押し返し……ということもありました）しているうちに年も明け，もう2月。ずいぶん遅くなってしまい，本当に申し訳なく思っています。

これだけ時間をかけたものの，まだまだ細かい点を見ればチェックしきれていない所や掘り下げが足りない部分もあることでしょう。私の技量不足というだけではなく，それだけ X 68000 は奥の深い機械ということでもあると思います。

グラフィックやサウンドなどの表に現れるような部分は，当然のことながら，X 68000 にはカタログスペックとして表れてこない部分まで執拗に追い続ける，マニアックなこだわりが随所に見られます。

電源スイッチだけでは電源が切れないようにしたり，電源が切れても，キーボードには電源が供給されているようにしてみたり，画面全体のコントラストを 16 段階に切り替えられるようにしようなどとは，ビジネスパソコン屋なら考えもつかないでしょう。FDD ひとつをとってみても，カタログスペックを優先させるなら，なにもオートイジェクトができる必要はありません。実際，FDD 業界は過当競争気味で，コストダウンが最優先であり，オートイジェクト機構などという余分な機能が付いた FDD はどこもつくりたがらないのです。コスト的にも，一般的なレバー付きのものを使ったほうが有利に決まっています。カタログ上にも出ないような部分であるとしても，気持ちよく使えるようになるのであれば，たとえコストアップになったとしても，あえてその道を選ぶ，そんな設計はビジネスパソコン屋にはまず不可能でしょう。

たとえカタログ上は他のマシンと同等かやや下に思えても，このような X 68000 の裏の部分までのこだわりが，長く使っていく間に本当の満足感となっていくのだろうと思います。

そんな X 68000 の潜在能力を引き出し，パーソナルコンピューティングの世界を目指すあなたに，本書がなにがしかの手助けとなれば，筆者としてこれに勝る喜びはありません。

1992 年 2 月 11 日（火）冬季オリンピックを CZ-600 DE で眺めつつ

桑野雅彦（くわの まさひこ）

参考文献

X 68000 テクニカルデータブック ……アスキー
X 68000 ベスト・プログラミング入門 ……技術評論社
半導体集積回路　通信用 LSI ……沖電気工業
マイクロコンピュータ技術資料　Z8530SCC ……シャープ
SCSI ボード　CZ-6BS1 取扱説明書……シャープ
16/32 ビットマイクロプロセッサ　TLCS-68000 周辺デバイス ……東芝
16 ビットマイクロプロセッサ　TLCS-68000 マイクロプロセッサ編 ……東芝
YM2151 カタログ……日本楽器製造
YM2151 アプリケーションカタログ……日本楽器製造
YM2608 アプリケーションマニュアル……日本楽器製造
μPD72065/72066 CMOS FDC ユーザーズマニュアル ……日本電気
DS300B-100/DS500B-100 ディスクコントローラ機能仕様書……日本電気
マルチチップ……日本電気
SEMICONDUCTOR DATA BOOK　8/16 ビットマイクロコンピュータ ……日立製作所
SCSI プロトコルコントローラ MB89352A ユーザーズマニュアル ……富士通
インテリジェントディスクコントローラ OEM マニュアル……富士通
RP5C15 ユーザーズマニュアル ……リコー
MOS Microprocessor and Peripherals ……AMD
最新 SCSI マニュアル ……CQ 出版社
M68000 マイクロプロセッサ　ユーザーズ・マニュアル ……CQ 出版社
MC68901 MULTI-FUNCTION PERIPHERAL (Advanced Information) ……MOTOROLA
MC68881 User's Manual ……MOTOROLA
ANSI X3.131-1986 ……American National Standard Institute
X 68000 技術資料 ……シャープ

INDEX●英数字順

INDEX●五十音順

Inside X68000

1992年4月23日　初版第1刷発行
1992年12月12日　初版第4刷発行

著　者　桑野雅彦（くわのまさひこ）
発行者　孫　正義
発行所　ソフトバンク株式会社　出版事業部
〒108　東京都港区高輪2-19-13　NS高輪ビル
営業部　☎03(5488)1360
編集部　☎03(5488)1326
印刷所　壮光舎印刷株式会社

ISBN4-89052-304-9 C0055

ISBN4-89052-304-9 C0055 P6800E

定価…6800円[本体・6602円]

本書は、シャープのX68000本体に内蔵されているCPUおよび周辺LSIの動作を、
すでに公開されている技術資料をもとに、筆者自身が実際に
動作確認しながら調べ上げたテクニカルデータブックです。
記述にあたっては、画面制御関連はいうまでもなく、
既存の資料にはほとんど記述されていない(あるいは、まったく記述されていない)
DMA、数値演算プロセッサ、FM音源、ADPCM、SASI、SCSIなどについて
詳細な記述が加えられています。
さらに、読者の方が動作確認できるように、gcc(XCでも可)を使った
サンプルプログラムも付いており、たいへん実践的な内容になっています。